ZHENZHONG XIAO

FROM PROBABILISTIC TO VERBALIZED REASONING AND MACHINE LEARNING

From Probabilistic to Verbalized Reasoning and Machine Learning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Zhenzhong Xiao

aus Guangdong, China

Tübingen

2026

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	23.07.2026
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Philipp Hennig
2. Berichterstatter:	Dr. Jonas Geiping

ABSTRACT

A fundamental question in machine learning is how to represent information and perform inference. Both tasks require reasoning under uncertainty: information must be encoded in representations that capture relevant structure, and inference must recover unobserved quantities from observed data. Probabilistic models address this through learned numerical representations with explicit distributional semantics. More recently, large language models have opened a complementary path: representing and processing information in natural language. From an information-theoretic perspective, language has been modeled as a probability distribution over sequences since Shannon's foundational work; natural language is thus inherently a probabilistic medium, carrying uncertainty through its ambiguity and context-dependence. This thesis investigates both paradigms and traces a path from one to the other.

In the first part, we develop tools for inference and learning in numerical representation spaces. We introduce an information-theoretic framework for hierarchical variational autoencoders that treats each latent layer as operating at a controllable point on a rate-distortion curve, enabling systematic trading of information between layers. We then investigate when these probabilistic representations generalize, decomposing VAE performance into generalization, amortization, and robustness gaps, and showing that synthetic data from diffusion models and overparameterization can improve training while also uncovering a double descent phenomenon in generative models.

In the second part, we turn to inference and learning in natural language space. We first establish conceptual foundations by drawing structural parallels between large language models and modern computers, arguing that LLMs constitute a new kind of general-purpose computing system where language serves as both program and data representation. Building on this, we introduce Verbalized Machine Learning, a framework in which models are parameterized entirely by natural language prompts and trained through iterative text-based optimization, demonstrating that learning can operate directly in language space. Finally, we address a key limitation: the knowledge-sampling gap, where LLMs can describe probability distributions but fail to sample from them faithfully. We propose Verbalized Rejection Sampling, which adapts classical rejection sampling to operate entirely in natural language, and provide theoretical bounds on when it outperforms direct sampling.

This last contribution bridges both parts of the thesis by showing that a classical probabilistic algorithm can be adapted to operate in language space while retaining theoretical guarantees. As large language models increasingly serve as general-purpose reasoning engines, verbalized computing, the practice of performing computation directly in natural language, is emerging as a new paradigm. Since natural language is inherently ambiguous, a central challenge for this paradigm is how to perform computation in it effectively and reliably. This thesis suggests that

the rich toolkit of probabilistic methods, which has been developed precisely for handling uncertainty, can provide both inspiration and principled foundations for addressing this challenge.

ZUSAMMENFASSUNG

Eine grundlegende Frage des maschinellen Lernens betrifft die Repräsentation von Informationen und die Durchführung von Inferenz. Beide Aufgaben erfordern Schlussfolgern unter Unsicherheit: Informationen müssen in Repräsentationen kodiert werden, die relevante Struktur erfassen, und Inferenz muss unbeobachtete Größen aus beobachteten Daten rekonstruieren. Probabilistische Modelle begegnen dieser Herausforderung durch gelernte numerische Repräsentationen mit expliziter Verteilungssemantik. In jüngerer Zeit haben große Sprachmodelle (LLMs) einen komplementären Weg eröffnet: die Repräsentation und Verarbeitung von Informationen in natürlicher Sprache. Aus informationstheoretischer Sicht wird Sprache seit Shannons grundlegender Arbeit als Wahrscheinlichkeitsverteilung über Sequenzen modelliert; natürliche Sprache ist somit von Natur aus ein probabilistisches Medium, das Unsicherheit durch Mehrdeutigkeit und Kontextabhängigkeit trägt. Die vorliegende Arbeit untersucht beide Paradigmen und verfolgt einen Weg von einem zum anderen.

Im ersten Teil entwickeln wir Werkzeuge für Inferenz und Lernen in numerischen Repräsentationsräumen. Wir stellen ein informationstheoretisches Rahmenwerk für hierarchische Variational Autoencoders (VAEs) vor, in dem jede latente Schicht als ein steuerbarer Punkt auf einer Rate-Distortion-Kurve behandelt wird, was einen systematischen Informationsaustausch zwischen den Schichten ermöglicht. Anschließend untersuchen wir, wann diese probabilistischen Repräsentationen generalisieren, zerlegen die Leistung von VAEs in Generalisierungs-, Amortisierungs- und Robustheitslücken und zeigen, dass synthetische Daten aus Diffusionsmodellen und Überparametrisierung das Training verbessern können, wobei wir zugleich ein Double-Descent-Phänomen in generativen Modellen aufdecken.

Im zweiten Teil wenden wir uns der Inferenz und dem Lernen im Raum der natürlichen Sprache zu. Zunächst schaffen wir konzeptionelle Grundlagen, indem wir strukturelle Parallelen zwischen großen Sprachmodellen und modernen Computern ziehen, und argumentieren, dass LLMs eine neue Art von Allzweck-Rechensystem darstellen, in dem Sprache sowohl als Programm als auch als Datenrepräsentation dient. Darauf aufbauend stellen wir Verbalized Machine Learning vor, ein Rahmenwerk, in dem Modelle vollständig durch Prompts in natürlicher Sprache parametrisiert und durch iterative textbasierte Optimierung trainiert werden, was zeigt, dass Lernen direkt im Sprachraum stattfinden kann. Schließlich adressieren wir eine wesentliche Einschränkung: die Wissens-Sampling-Lücke, bei der LLMs Wahrscheinlichkeitsverteilungen beschreiben können, jedoch nicht in der Lage sind, treu aus ihnen Stichproben zu ziehen. Wir schlagen Verbalized Rejection Sampling vor, das klassisches Rejection Sampling so adaptiert, dass es vollständig in natürlicher Sprache arbeitet, und geben theoretische Schranken dafür an, wann es direktes Sampling übertrifft.

Dieser letzte Beitrag verbindet beide Teile der Arbeit, indem er zeigt, dass ein klassischer probabilistischer Algorithmus so adaptiert werden kann, dass er im Sprachraum arbeitet und dabei theoretische Garantien behält. Da große Sprachmodelle zunehmend als allgemeine Reasoning-Engines dienen, entwickelt sich verbalized computing, die Praxis des Rechnens direkt in natürlicher Sprache, zu einem neuen Paradigma. Da natürliche Sprache von Natur aus mehrdeutig ist, besteht eine zentrale Herausforderung dieses Paradigmas darin, wie in ihr Berechnungen effektiv und zuverlässig durchgeführt werden können. Die vorliegende Arbeit legt nahe, dass der reichhaltige Werkzeugkasten probabilistischer Methoden, der gerade zum Umgang mit Unsicherheit entwickelt wurde, sowohl Inspiration als auch fundierte Grundlagen zur Bewältigung dieser Herausforderung bieten kann.

AUTHOR CONTRIBUTIONS

All text in the main part of this thesis was written by myself.

Some results presented in this thesis were created in joint work between me and co-authors of the publications that are included with this thesis. For detailed author contributions, see Appendix B.

CONTENTS

1

INTRODUCTION

Two threads run through the history of computing. The first asks what computers should ultimately be able to do: Turing envisioned machines that think, learn, and converse in natural language. The second asks how to make information processing reliable: Shannon and von Neumann showed that probability theory is the key. This thesis sits at the point where these threads converge. Large language models have brought computation into natural language, approaching Turing's aspiration, but language is a semantically unreliable medium. The five papers collected here argue that the probabilistic toolkit, the same tradition that made communication and hardware reliable, provides both inspiration and principled foundations for making computation in language reliable too.

1.1 FROM SYMBOLS TO NUMBERS TO LANGUAGE

Turing gave computer science two things. The first was a precise definition of what is computable: the universal machine, an abstract device that manipulates symbols according to formal rules [1]. Any procedure that can be described unambiguously can, in principle, be carried out by such a machine. The second was an aspiration. In 1950, Turing asked whether machines could think, and proposed that the test should be a conversation: if a machine's responses are indistinguishable from a human's, we have no grounds to deny its intelligence [2]. He envisioned machines that learn from experience, that reason, that communicate in natural language. The theoretical foundation was general-purpose. The aspiration was linguistic.

Meanwhile, a separate line of work asked a different question: not what to compute, but how to compute reliably. Shannon solved what had seemed an impossible problem: how to transmit messages over a channel that corrupts them [3]. The key insight was that redundancy, structured by probability theory, can tame noise. By characterizing the channel's capacity, Shannon showed that the boundary between achievable and impossible could be drawn exactly, and that codes approaching this limit exist. Von Neumann, directly inspired by Shannon, asked the same question about hardware: how can a machine built from unreliable components produce reliable results [4]? His answer was the same in spirit. Redundancy in circuit design can achieve arbitrarily low error rates, provided each component's failure probability stays below a threshold. In both cases, probability theory was the tool, and a threshold theorem told us exactly when reliability is achievable.

Probability theory expanded beyond reliability to inference and learning under uncertainty. The resulting toolkit includes variational inference, sampling algorithms, information-theoretic objectives, and deep generative models. Section 1.2 describes this toolkit in more detail. Turing's aspiration of linguistic intelligence, meanwhile,

was pursued separately through symbolic AI, with limited success. Large language models brought linguistic reasoning within computational reach [5]. They got there not through symbolic AI but through the probabilistic numerical tradition: neural networks trained on massive text corpora. Language re-entered computation not by replacing the numerical toolkit but by being absorbed into it.

Yet language brings a new kind of noise. Shannon and von Neumann faced physical unreliability, operating at the level of signals and circuits: bit flips, signal attenuation, faulty vacuum tubes. The noise in language is *semantic*: ambiguity rather than bit flips, hallucination rather than hardware failure, context-dependence rather than thermal fluctuation. An LLM asked to summarize a legal contract may produce fluent, confident text that misrepresents a key clause. The "noise" lives in the space of meaning, not in the physical channel. So the old question returns at a higher level of abstraction: *how do you compute reliably with a semantically unreliable medium?*

The same pattern, we argue, applies. The probabilistic toolkit developed for numerical spaces, the same intellectual tradition that descends from Shannon and von Neumann, provides both inspiration and principled foundations for making computation in language reliable. The shift from physical to semantic noise makes this a harder problem: semantic unreliability is not yet characterized mathematically in the way that signal noise and hardware failure are. But the approach is likely similar. We call this emerging paradigm *verbalized computing*. Whether analogues of Shannon's channel capacity or von Neumann's threshold theorem exist for verbalized computing, giving precise limits on when language-based inference can be made reliable and at what cost, remains an open question. This thesis does not answer it, but the contributions presented here take first steps, and Chapter 5 returns to this question as a horizon for future work.

1.2 TWO PARADIGMS: NUMERICAL SPACES AND NATURAL LANGUAGE

Before turning to the specific contributions of this thesis, we sketch the two paradigms in more detail: the probabilistic toolkit that emerged from the numerical tradition, and the nascent practice of reasoning in natural language.

PROBABILISTIC REASONING IN NUMERICAL SPACES. The first paradigm encodes information as points in continuous numerical spaces and uses the calculus of probability to perform inference. A generative model posits a joint distribution over observed data $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$, and inference amounts to computing the posterior $p(\boldsymbol{z} \mid \boldsymbol{x})$: given what we observed, what do we believe about the hidden variables [6]? This is the classical inverse probability problem. Because the posterior is rarely available in closed form, practical methods resort to approximations. Variational inference replaces the exact posterior with a tractable family and optimizes a bound on the marginal likelihood [7, 8]. Sampling methods such as Markov chain Monte Carlo construct sequences of random states that converge to the posterior [9].

Within this paradigm, information theory provides both a unifying language and a design principle. The evidence lower bound (ELBO) that variational autoencoders (VAEs) optimize can be decomposed into a reconstruction term and a rate term, revealing a direct connection to lossy source coding [10]. Tuning this rate–distortion trade-off controls how much information the latent representation retains about the input, and allows the representation to be tailored for different downstream tasks. A separate but equally important question is generalization: how well do these models perform on data they have not seen during training? This depends on subtle interactions between model capacity, training data, and the inference procedure [11], and remains a central challenge.

VERBALIZED REASONING IN NATURAL LANGUAGE. The second paradigm represents information not as numerical vectors but as sequences of natural language tokens. Large language models, trained to predict the next token in a sequence, have emerged as surprisingly general-purpose problem solvers. By scaling model size and training data, LLMs acquire the ability to perform tasks they were never explicitly trained for, from translation and summarization to mathematical reasoning and code generation [5, 12]. This ability was not predicted by smaller scale models and has prompted a re-examination of what it means to "program" a machine.

A striking structural parallel exists between LLMs and stored-program computers. Just as von Neumann's architecture unifies programs and data in the same memory, an LLM unifies instructions, context, and outputs in the same token space. A text prompt serves as a "program" that specifies what the model should compute, while the model's response serves as the "output." This parallel suggests treating natural language not merely as a user interface but as a genuine medium of computation. Within this framing, the classical tasks of machine learning, including learning, optimization, and sampling, can in principle be revisited in language space. Models can be parameterized by natural language descriptions rather than numerical weight vectors; training can proceed through iterative text-based refinement rather than gradient descent; and inference procedures can operate through verbal reasoning rather than numerical optimization.

CONTRASTS AND CONNECTIONS. The central tension between the two paradigms is computational. In numerical spaces, the representation is continuous and differentiable, so inference can exploit gradient-based optimization with well understood convergence guarantees. Yet the resulting latent codes are opaque, and the computations they support are tightly coupled to the model architecture. In language space, the representation is discrete and compositional, so inference operates through flexible, human readable reasoning steps. Yet this flexibility comes at the cost of formal guarantees, and the theoretical properties of LLM-mediated computation are only beginning to be understood. Natural language is, in this sense, an unreliable computational medium: expressive enough to represent almost any reasoning step, but lacking the mathematical discipline that makes numerical methods trustworthy.

Yet beneath this tension lies a shared probabilistic foundation. A language model defines a probability distribution over token sequences, and generating text from an LLM is, mathematically, a sampling procedure. When an LLM is prompted to classify, predict, or reason, it is implicitly performing approximate inference over a distribution shaped by its training data and the prompt. This shared foundation suggests that the computational tools developed for numerical spaces, from variational bounds and information-theoretic objectives to sampling algorithms, have a natural place in language space as well. How to make computation in this medium possible and reliable, and how far the probabilistic toolkit can take us, are the questions this thesis takes first steps on.

1.3 CONTRIBUTIONS OF THIS THESIS

The five papers in this thesis trace a path from probabilistic reasoning in numerical spaces to verbalized reasoning in natural language. The first two contributions develop and refine the probabilistic toolkit in its home territory; the remaining three carry core ideas from that toolkit, specifically representation, learning, and sampling, into the new and unreliable medium of language.

STRENGTHENING THE PROBABILISTIC TOOLKIT. The first contribution develops an information-theoretic framework for hierarchical variational autoencoders (VAEs) that enables independent control of each layer's rate [13]. By decomposing the evidence lower bound into per-layer rate and distortion terms, the framework reveals how optimal information allocation across latent layers depends on the downstream task, whether reconstruction, representation learning, or generation. This work extends the classical rate–distortion perspective from single-layer to hierarchical models, providing both theoretical bounds and large-scale empirical validation. The second contribution investigates generalization in VAEs from a deep learning perspective, studying the effects of synthetic data from diffusion models and overparameterization on the generalization, amortization, and robustness gaps [14]. Together, these two works deepen our understanding of how to represent information in learned latent spaces and how to ensure that the resulting models generalize. These are foundational questions for any probabilistic approach, and the tools developed here inform the second half of the thesis.

CARRYING PROBABILISTIC TOOLS TO NATURAL LANGUAGE. The conceptual foundations for the second half of the thesis are laid by drawing a structural parallel between LLMs and stored-program computers [15]. Just as von Neumann's architecture unifies programs and data in the same memory, an LLM unifies instructions, context, and outputs in the same token space. This parallel motivates treating natural language as a genuine computational medium and provides the conceptual vocabulary for what follows. Building on this foundation, the verbalized machine learning (VML) framework demonstrates that classical learning tasks can be solved entirely in language space, with models parameterized by text prompts

and trained through iterative verbalized optimization [16]. VML shows that the idea of "learning" transfers from numerical to natural language representations, though the mechanisms and trade-offs differ fundamentally. Finally, verbalized rejection sampling (VRS) transplants rejection sampling, a classical probabilistic algorithm, into language space [17]. VRS shows that LLMs can generate less biased samples when guided by a verbalized accept/reject procedure, and provides theoretical guarantees for when this approach improves over direct sampling. Of the five contributions, VRS is the most direct application of a classical probabilistic algorithm to the language medium, bringing algorithmic guarantees of a kind that prompt engineering alone cannot supply.

Taken together, these five papers argue that the probabilistic toolkit provides both inspiration and principled foundations for verbalized computing. Chapter 5 develops this argument and discusses its broader implications, including the role of formalization complexity in reasoning about when verbalized approaches are worth considering.

1.4 THESIS OUTLINE

The remainder of this thesis is organized as follows.

Chapter 2 provides the technical background necessary for the contributions that follow, covering information theory and Bayesian inference, variational inference and latent variable models, language models with their prompting and in-context learning capabilities, and the structural connections that link the two paradigms.

Chapter 3 presents two contributions to probabilistic reasoning and learning in numerical spaces. First, we develop an information-theoretic framework for hierarchical VAEs that enables independent control of each layer's rate, with theoretical bounds and large-scale experiments showing how optimal information allocation depends on the downstream task. Second, we investigate generalization in VAEs from a deep learning perspective, studying the effects of synthetic data from diffusion models and overparameterization on the generalization, amortization, and robustness gaps.

Chapter 4 presents three contributions to verbalized reasoning and machine learning. First, we establish conceptual foundations by drawing structural parallels between LLMs and stored-program computers. Second, we introduce the VML framework, where machine learning models are parameterized entirely by natural language and trained through iterative text-based optimization. Third, we adapt classical rejection sampling to language space, providing both empirical evidence and theoretical analysis showing that this verbalized procedure reduces sampling bias.

Chapter 5 reflects on the broader implications of the work through the lens of formalization complexity, and outlines future directions for verbalized computing.

The Appendix includes all publications and outlines author contributions to each of these publications. Below is a list of the five publications included with this thesis,

as well as an additional set of 9 publications to which I contributed throughout my PhD, but are not included in this thesis.

List of publications included in this thesis

The following publications are included in this thesis:

Tim Z. Xiao, Robert Bamler (2023), Trading Information between Latents in Hierarchical Variational Autoencoders, *International Conference on Learning Representations (ICLR)* [13]

Tim Z. Xiao*, Johannes Zenn*, Robert Bamler (2024), A Note on Generalization in Variational Autoencoders: How Effective Is Synthetic Data and Overparameterization?, *Transactions on Machine Learning Research (TMLR)* [14]

Tim Z. Xiao, Weiyang Liu, Robert Bamler (2025), Large Language Models Are Zero-Shot Problem Solvers — Just Like Modern Computers, *Harvard Data Science Review* [15]

Tim Z. Xiao, Robert Bamler, Bernhard Schölkopf, Weiyang Liu (2025), Verbalized Machine Learning: Revisiting Machine Learning with Language Models, *Transactions on Machine Learning Research (TMLR)* [16]

Tim Z. Xiao, Johannes Zenn, Zhen Liu, Weiyang Liu, Robert Bamler, Bernhard Schölkopf (2025), Flipping Against All Odds: Reducing LLM Coin Flip Bias via Verbalized Rejection Sampling, *arXiv preprint* [17]

List of publications not included in this thesis

I contributed to the following publications throughout my PhD, but they are not included in this thesis:

Zeju Qiu, Simon Buchholz, **Tim Z. Xiao**, Maximilian Dax, Bernhard Schölkopf, Weiyang Liu (2025), Reparameterized LLM Training via Orthogonal Equivalence Transformation, *NeurIPS* [18]

Zeju Qiu*, Weiyang Liu*, Haiwen Feng*, Zhen Liu, **Tim Z. Xiao**, Katherine M. Collins, Joshua B. Tenenbaum, Adrian Weller, Michael J. Black, Bernhard Schölkopf (2025), Can Large Language Models Understand Symbolic Graphics Programs?, *ICLR* [19]

Zhen Liu, **Tim Z. Xiao***, Weiyang Liu*, Yoshua Bengio, Dinghuai Zhang (2025), Fast Diversity-Preserving Reward Finetuning of Diffusion Models via Nabla-GFlowNets, *ICLR* [20]

Zijing Ou*, Mingtian Zhang*, Andi Zhang, **Tim Z. Xiao**, Yingzhen Li, David Barber (2025), Improving Probabilistic Diffusion Models With Optimal Covariance Matching, *ICLR* [21]

Andi Zhang, **Tim Z. Xiao**, Weiyang Liu, Robert Bamler, Damon Wischik (2025), Your Finetuned Large Language Model is Already a Powerful Out-of-distribution Detector, *AISTATS* [22]

Zhouliang Yu*, Yuhuan Yuan*, **Tim Z. Xiao**, Frank Xia, Jie Fu, Ge Zhang, Ge Lin, Weiyang Liu (2025), Generating Symbolic World Models via Test-time Scaling of Large Language Models, *TMLR* [23]

Tim Z. Xiao, Weiyang Liu, Robert Bamler (2023), A Compact Representation for Bayesian Neural Networks By Removing Permutation Symmetry, *NeurIPS 2023 Workshop on Unifying Representations in Neural Models* [24]

Tim Z. Xiao*, Johannes Zenn*, Robert Bamler (2023), The SVHN Dataset Is Deceptive for Probabilistic Generative Models Due to a Distribution Mismatch, *NeurIPS 2023 Workshop on Distribution Shifts* [25]

Zeju Qiu*, Weiyang Liu*, **Tim Z. Xiao**, Zhen Liu, Umang Bhatt, Yucen Luo, Adrian Weller, Bernhard Schölkopf (2023), Iterative Teaching by Data Hallucination, *AISTATS* [26]

* indicates equal contribution.

2

BACKGROUND

This chapter provides the technical background for the contributions that follow. We begin with information theory and Bayesian inference (Section 2.1), then develop variational and sampling methods for latent variable models (Section 2.2). Section 2.3 introduces language models and the prompting capabilities that make them general-purpose computational tools. Finally, Section 2.4 draws out the structural connections between the two paradigms and identifies the gap that the rest of the thesis fills.

2.1 INFORMATION THEORY AND PROBABILISTIC INFERENCE

Much of science and engineering can be organized around two kinds of problem. The *forward problem* is: given a model with parameters $\boldsymbol{\theta}$, what data $\boldsymbol{x}$ would we expect to see? We write the answer as a probability distribution $p(\boldsymbol{x} \mid \boldsymbol{\theta})$, which assigns a probability to every possible dataset under the assumed model. Forward problems are, in principle, straightforward: specify the mechanism, turn the crank, and read off predictions. The *inverse problem* reverses the arrow: given observed data $\boldsymbol{x}$, what can we infer about the parameters $\boldsymbol{\theta}$ that generated it? This is almost always the harder direction. Data are noisy, models are approximate, and many different parameter settings may be consistent with the same observations. To solve inverse problems we need a language for quantifying uncertainty, for measuring how much a piece of data tells us, and for updating beliefs in light of evidence. Information theory and probability theory, developed together, provide exactly this language.

This section introduces the core concepts in that language. We begin with entropy and information content, which measure the uncertainty associated with a random variable. We then define the Kullback–Leibler divergence, which measures the cost of using the wrong distribution, and mutual information, which measures statistical dependence. Finally, we arrive at Bayes' theorem, the fundamental tool for solving inverse problems, and identify the computational bottleneck that motivates the approximation methods of Section 2.2. We write log throughout; the base is 2 (bits) when the context is information-theoretic and e (nats) when the context is variational inference or deep learning, following the conventions of each field.

INFORMATION CONTENT AND ENTROPY. Consider a discrete random variable X with outcomes x drawn from an alphabet $\mathcal{A}_X$ according to $p(x)$. When an outcome x occurs, the *Shannon information content* of that outcome is

$$h(x) = \log \frac{1}{p(x)}, \tag{2.1}$$

measured in bits. The definition captures a simple intuition: rare events are more surprising and therefore more informative. An outcome that occurs with certainty ($p(x) = 1$) carries no information; an outcome that occurs with probability 10^{-6} carries about 20 bits.

The *entropy* of the random variable X is the expected information content,

$$H[X] = \mathbb{E}_{p(x)}\left[\log \frac{1}{p(x)}\right] = -\sum_{x \in \mathcal{A}_X} p(x) \log p(x)\,. \tag{2.2}$$

Entropy measures the average surprise we experience when observing outcomes from p. It is non-negative, equal to zero only when X is deterministic, and maximized when X is uniformly distributed over $\mathcal{A}_X$. A concrete example will recur throughout this thesis: a coin with probability p of landing heads has entropy $H = -p \log p - (1-p)\log(1-p)$. A fair coin ($p = 1/2$) has $H = 1$ bit, which is the maximum; the entropy vanishes at $p = 0$ or $p = 1$ (certainty). The same Bernoulli setting reappears in Chapter 4, where the gap between what a language model "knows" about a coin's bias and what it produces when asked to flip that coin motivates a verbalized correction procedure.

For continuous random variables $\boldsymbol{x}$ with density $p(\boldsymbol{x})$, the sum becomes an integral and the resulting quantity is called the *differential entropy*,

$$h[\boldsymbol{x}] = -\int p(\boldsymbol{x}) \log p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}\,. \tag{2.3}$$

Differential entropy can be negative (a Gaussian with very small variance, for instance, has negative differential entropy), but the differences and divergences we build from it remain well behaved [27].

KULLBACK–LEIBLER DIVERGENCE. Entropy tells us about a single distribution. To compare two distributions we need a divergence. Suppose we model a source distributed according to p using a code optimized for a different distribution q. The expected extra cost per symbol is the *Kullback–Leibler divergence*:

$$D_{\mathrm{KL}}(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = \mathbb{E}_p\left[\log \frac{p(x)}{q(x)}\right]. \tag{2.4}$$

Three properties are essential. First, $D_{\mathrm{KL}}(p\|q) \geq 0$ for all p and q, with equality if and only if $p = q$ almost everywhere. This follows from Jensen's inequality applied to the concave logarithm [27]. Second, the KL divergence is *not* symmetric: $D_{\mathrm{KL}}(p\|q) \neq D_{\mathrm{KL}}(q\|p)$ in general. This asymmetry has practical consequences. Minimizing $D_{\mathrm{KL}}(q\|p)$ over q produces an approximation that avoids placing mass where p is small, tending to lock onto a single mode of p. Minimizing $D_{\mathrm{KL}}(p\|q)$ produces an approximation that covers all of p's mass, even at the cost of spreading probability into regions where p is negligible. Variational inference, as we will see in Section 2.2, minimizes the first of these two directions, which is why variational approximations tend to be more compact than the true posterior they approximate.

Third, $D_{\mathrm{KL}}(p\|q)$ is undefined (or infinite) whenever $q(x) = 0$ for some x where $p(x) > 0$: the approximation must have support wherever the target has support.

Closely related is the *cross-entropy* between p and q:

$$H(p, q) = -\sum_x p(x) \log q(x) = H[p] + D_{\mathrm{KL}}(p\|q)\,. \tag{2.5}$$

Since $D_{\mathrm{KL}}(p\|q) \geq 0$, the cross-entropy is always at least as large as the entropy. This decomposition will reappear in Section 2.3: the standard training objective for language models is the cross-entropy between the data distribution and the model, which can be read as the entropy of the data (an irreducible floor) plus the KL divergence from the data to the model (the part training can reduce) [28].

MUTUAL INFORMATION. When we have two random variables, X and Z, we often want to know how much observing one tells us about the other. The *mutual information* is defined as the KL divergence between the joint distribution and the product of the marginals:

$$I(X; Z) = D_{\mathrm{KL}}\big(p(x, z) \,\|\, p(x)\, p(z)\big) = H[X] - H[X \mid Z] = H[Z] - H[Z \mid X]\,, \tag{2.6}$$

where $H[X \mid Z] = \mathbb{E}_{p(z)}[H[X \mid Z = z]]$ is the conditional entropy. Mutual information is symmetric in X and Z, non-negative, and zero if and only if X and Z are independent. It measures the reduction in uncertainty about X that comes from observing Z, or equivalently, the reduction in uncertainty about Z from observing X.

A fundamental constraint on mutual information is the *data processing inequality*: if X, Z, and W form a Markov chain $X \to Z \to W$, then $I(X; W) \leq I(X; Z)$ [27]. Processing can only destroy information, never create it. This inequality bounds how much any downstream representation can know about the original input and underlies the information bottleneck perspective on representation learning. In particular, the hierarchical information trading framework developed in Chapter 3 decomposes the mutual information between data and latent variables across the layers of a hierarchical model, showing that different downstream tasks demand different allocations of information across layers.

BAYES' THEOREM AND THE INVERSE PROBLEM. We now return to the inverse problem. Given observed data $\boldsymbol{x}$ and a model with parameters $\boldsymbol{\theta}$, Bayes' theorem tells us how to update our beliefs:

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\boldsymbol{x})}\,. \tag{2.7}$$

Each term has a name and a role. The *prior* $p(\boldsymbol{\theta})$ encodes what we believe about the parameters before seeing any data. The *likelihood* $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ measures how well each parameter setting explains the observed data: it is the forward model, evaluated at $\boldsymbol{x}$. The *posterior* $p(\boldsymbol{\theta} \mid \boldsymbol{x})$ is what we want: our updated belief after the data have

arrived. In words, what we know about $\boldsymbol{\theta}$ after observing $\boldsymbol{x}$ is determined by what we knew before, multiplied by what the data told us.

The denominator,

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}\,, \tag{2.8}$$

is called the *evidence* or marginal likelihood. It plays two distinct roles. First, it normalizes the posterior, ensuring that $p(\boldsymbol{\theta} \mid \boldsymbol{x})$ integrates to one. Second, and less obviously, it provides a principled score for comparing different models: a model $\mathcal{H}$ that assigns high evidence $p(\boldsymbol{x} \mid \mathcal{H})$ to the observed data is one that predicted those data well, averaged over its prior. This is the Bayesian form of Occam's razor: complex models spread their prior mass thinly over many possible datasets and therefore assign lower evidence to any single dataset, while simple models concentrate their predictions and are rewarded when those predictions match the data [6]. This two-level structure, first inferring parameters within a model and then comparing models via their evidence, organizes much of Bayesian reasoning [6].

The computational difficulty of Bayesian inference lies almost entirely in the evidence integral in Equation (2.8). For models with many parameters, the integral is high-dimensional and rarely admits a closed-form solution. This intractability is not a failure of the framework but a reflection of the difficulty of the inverse problem itself: the space of possible explanations is vast, and summarizing it faithfully requires evaluating the likelihood across that entire space. Two families of approximation have been developed to address this bottleneck. *Variational methods* replace the exact posterior with a tractable family of distributions and optimize a divergence measure, converting the integration problem into an optimization problem. *Sampling methods* construct sequences of random draws that converge to the posterior, either as dependent chains (Markov chain Monte Carlo) or as independent draws from a simpler proposal distribution (rejection sampling). Section 2.2 develops both approaches, focusing on the specific tools needed for the contributions in Chapters 3 and 4.

2.2 VARIATIONAL INFERENCE AND LATENT VARIABLE MODELS

Section 2.1 ended with a problem: the evidence integral that makes Bayesian inference exact is almost never tractable. This section develops the probabilistic tools most relevant to this thesis. Section 2.2.1 introduces variational inference, which turns the integration problem into an optimization problem and leads to variational autoencoders and their hierarchical extensions. Section 2.2.2 then covers rejection sampling, a classical method for drawing exact samples from a target distribution using a simpler proposal, which will reappear in a new form in Chapter 4.

2.2.1 *Latent Variable Models and the Evidence Lower Bound*

A latent variable model posits that the observed data $\boldsymbol{x}$ were generated by a two-step process: first, a set of hidden variables $\boldsymbol{z}$ is drawn from a prior $p(\boldsymbol{z})$; then the data

are drawn from a conditional distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ parameterized by θ. The latent variables represent the hidden causes or structure underlying the observations, and the generative model $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})$ defines a forward problem in the sense of Section 2.1. The corresponding inverse problem is posterior inference: given an observation $\boldsymbol{x}$, compute $p(\boldsymbol{z} \mid \boldsymbol{x})$, the distribution over latent variables that could have produced it. By Bayes' theorem in Equation (2.7), this requires the evidence $p(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}$, which is intractable for all but the simplest models because the integral ranges over the entire latent space.

Variational inference addresses this intractability by introducing an approximate posterior $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, a tractable family of distributions parameterized by ϕ, and optimizing ϕ so that q_ϕ is close to the true posterior. In the variational autoencoder (VAE) framework [8, 29], both the generative model (the *decoder*, parameterized by θ) and the approximate posterior (the *encoder*, parameterized by ϕ) are neural networks, and they are trained jointly. The encoder performs *amortized inference*: rather than optimizing variational parameters separately for each data point, a single network $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ maps any input $\boldsymbol{x}$ to an approximate posterior in a single forward pass. This amortization is what makes VAEs scalable to large datasets, but it comes at a cost: the encoder must approximate the optimal posterior for all inputs simultaneously, which introduces systematic errors that we return to below.

THE EVIDENCE LOWER BOUND. The objective that makes joint training of encoder and decoder possible is the *evidence lower bound* (ELBO). It can be derived by decomposing the log-evidence:

$$\log p(\boldsymbol{x}) = \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})] - D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p(\boldsymbol{z})\big)}_{\mathrm{ELBO}(\boldsymbol{x};\theta,\phi)} + \underbrace{D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p(\boldsymbol{z} \mid \boldsymbol{x})\big)}_{\geq 0}. \tag{2.9}$$

Because the last term is a KL divergence and therefore non-negative, the first two terms together form a lower bound on $\log p(\boldsymbol{x})$. The bound is tight if and only if $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = p(\boldsymbol{z} \mid \boldsymbol{x})$, that is, when the approximate posterior matches the true one. Maximizing the ELBO with respect to θ and ϕ simultaneously pushes the generative model toward higher likelihood and the encoder toward the true posterior.

The two terms in the ELBO have complementary roles. The first, $\mathbb{E}_{q_\phi}[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})]$, is the expected log-likelihood of the data under the decoder, evaluated at latent codes drawn from the encoder; it rewards accurate reconstruction. The second, $D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p(\boldsymbol{z}))$, penalizes the encoder for deviating from the prior; it acts as a regularizer that keeps the latent representations structured. To optimize the ELBO by gradient descent, gradients must flow through the sampling step $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$. The *reparameterization trick* makes this possible: one writes $\boldsymbol{z} = g(\boldsymbol{\epsilon}, \boldsymbol{x}; \phi)$ with $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ a fixed noise distribution (typically standard Gaussian), so that the randomness is decoupled from the parameters ϕ and standard back propagation applies [8, 29].

THE RATE-DISTORTION PERSPECTIVE. The ELBO can be rewritten as a trade-off between two information-theoretic quantities [10]. Define the *rate* and *distortion* as

$$R = D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p(\boldsymbol{z})\big)\,, \qquad D = -\,\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})]\,, \tag{2.10}$$

so that ELBO $= -D - R$ and maximizing the ELBO is equivalent to minimizing $D + R$. The rate R measures how many nats the latent code uses beyond what the prior provides for free; the distortion D measures how much reconstruction quality is lost. This is a direct connection to Shannon's rate-distortion theory: the encoder compresses the input into a latent code, and the decoder reconstructs from that code, with R and D playing exactly the roles of rate and distortion in lossy source coding.

The β-VAE [30] generalizes this trade-off by introducing a Lagrange multiplier β:

$$\mathcal{L}_\beta = D + \beta\, R\,. \tag{2.11}$$

The standard ELBO corresponds to $\beta = 1$. Setting $\beta > 1$ encourages stronger compression, pushing the latent representation toward the prior and potentially promoting disentangled features at the cost of reconstruction quality. Setting $\beta < 1$ permits higher rate and better reconstruction but less structured representations. The parameter β thus traces out a frontier in the rate-distortion plane, and the choice of operating point depends on the downstream task.

HIERARCHICAL VAES. A single layer of latent variables may not suffice to capture the multi-scale structure present in complex data. Hierarchical VAEs address this by introducing L layers of latent variables $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$, where lower layers capture fine-grained detail and higher layers capture global structure. The generative model factorizes top-down:

$$p_\theta(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_L) = p_\theta(\boldsymbol{z}_L) \prod_{\ell=1}^{L-1} p_\theta(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell}) \cdot p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_{\geq 1})\,, \tag{2.12}$$

where $\boldsymbol{z}_{>\ell} = (\boldsymbol{z}_{\ell+1}, \ldots, \boldsymbol{z}_L)$ denotes all layers above ℓ, and $\boldsymbol{z}_{\geq\ell} = (\boldsymbol{z}_\ell, \ldots, \boldsymbol{z}_L)$ denotes all layers including ℓ and above. The inference model is a joint distribution $q_\phi(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L \mid \boldsymbol{x})$, which can be factorized by the chain rule as $q_\phi(\boldsymbol{z}_L \mid \boldsymbol{x}) \prod_{\ell=1}^{L-1} q_\phi(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell}, \boldsymbol{x})$. Different inference architectures (e.g., the Ladder VAE [31], BIVA [32], NVAE [33], and Very Deep VAEs [34]) parameterize these conditionals in different ways.

Among these, certain architectures induce a factorization under which the total rate decomposes cleanly into per-layer contributions, as Section 3.1 develops:

$$R = \sum_{\ell=1}^{L} R_\ell\,, \qquad \text{where} \quad R_\ell = \mathbb{E}_{q_\phi(\boldsymbol{z}_{>\ell}|\boldsymbol{x})}\big[D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell}, \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell})\big)\big] \tag{2.13}$$

is the rate consumed by layer ℓ, averaged over the layers above it. This decomposition reveals that a single scalar β is too coarse: different downstream tasks may require different allocations of information across layers. The generalized objective

$$\mathcal{L}_{\boldsymbol{\beta}} = D + \sum_{\ell=1}^{L} \beta_\ell \, R_\ell \tag{2.14}$$

introduces a per-layer Lagrange multiplier β_ℓ, enabling independent control of each layer's rate. Reconstruction benefits from high total rate distributed across all layers; generation benefits from concentrating rate in the top layers that capture global structure; representation learning requires a task-dependent balance. Chapter 3 develops this framework in detail, providing theoretical bounds on optimal allocations and large-scale experiments demonstrating that no single set of β_ℓ values is optimal for all tasks.

DIFFUSION MODELS. VAEs and their hierarchical extensions are not the only class of deep generative models. *Diffusion models* [35, 36] learn to generate samples by reversing a gradual noising process: a forward process incrementally corrupts data with Gaussian noise until only noise remains, and a neural network is trained to reverse this process step by step, denoising pure noise back into a sample from the data distribution. Although the underlying mathematical machinery differs from that of VAEs, the goal is the same: to learn an approximation of the data distribution that supports sampling. Diffusion models have produced state-of-the-art generative quality on images and have become a popular source of synthetic training data, a property that Chapter 3 exploits when studying generalization in VAEs.

GENERALIZATION IN VAES. The ELBO is optimized on a finite training set $\mathcal{D}_{\text{train}}$, but we ultimately care about performance on unseen data. Three distinct gaps characterize how a trained VAE falls short of its ideal performance.

The *generalization gap* G_g measures overfitting: it is the difference in average ELBO between training and test data. A large G_g indicates that the model has memorized training examples rather than learning the underlying distribution. The *amortization gap* G_a measures the price of amortization: it is the difference between the ELBO achieved by the learned encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ and the ELBO that would be achieved by optimizing variational parameters individually for each test point [37]. Because a single encoder network must serve all inputs, it cannot match per-sample optimization, and $G_a \geq 0$ by construction. The *robustness gap* G_r measures sensitivity to perturbations: an encoder that has overfit to training data may map similar inputs to very different latent codes, making the representation brittle.

These three gaps depend on model capacity, training set size, and the inference architecture, and their interactions can be counterintuitive: classical learning theory warns against overparameterization, yet modern deep learning has shown that larger models can generalize better, a phenomenon known as double descent [38, 39]. Whether similar effects hold for VAEs, and how the three gaps respond to changes in model size and training data, is the subject of Chapter 3.

2.2.2 *Sampling and Rejection Sampling*

Variational inference converts the posterior computation into an optimization problem, but another family of methods tackles it directly: generate samples from the target distribution. Given a distribution $P(x)$ that may be known only up to a normalizing constant, the goal is to draw independent samples from P. If we can do so, expectations under P can be estimated by simple averaging.

REJECTION SAMPLING. Rejection sampling solves this problem exactly, using a simpler *proposal* distribution from which we can already sample. Suppose we wish to sample from a target $P(x)$ that may be known only up to a constant: $P(x) = P^*(x)/Z$. Given a proposal distribution $Q(x)$ and a constant M such that $M \cdot Q(x) \geq P^*(x)$ for all x, the algorithm proceeds as follows: draw a candidate $x \sim Q(x)$, then accept it with probability

$$A(x) = \frac{P^*(x)}{M \cdot Q(x)}\,. \tag{2.15}$$

If rejected, discard x and repeat. The accepted samples are distributed exactly according to P, because the probability of proposing and accepting a value at x is proportional to $Q(x) \cdot P^*(x)/(M \cdot Q(x)) = P^*(x)/M \propto P(x)$ [6, 40]. The overall acceptance rate is $\alpha = Z/M$; when P is already normalized ($Z = 1$), this simplifies to $\alpha = 1/M$.

The method is particularly clean in low-dimensional settings. For a Bernoulli target with parameter p and a uniform proposal $Q = \text{Bernoulli}(1/2)$, the envelope constant is $M = \max\{p/(1/2), (1-p)/(1/2)\} = 2\max\{p, 1-p\} \leq 2$, so the acceptance rate is at least $1/2$: at least half of all proposals are accepted. This Bernoulli case will serve as the testbed in Chapter 4, where candidate samples are drawn from a fixed proposal and a language model decides whether to accept or reject each one based on textual descriptions of the target and proposal distributions. The key finding is that this indirection, routing sampling through a binary accept-reject decision rather than asking the model to produce samples directly, substantially reduces sampling bias.

TOTAL VARIATION DISTANCE. To measure how close a sampling procedure's output is to the target distribution, we use the *total variation distance*:

$$D_{\text{TV}}(P_1, P_2) = \frac{1}{2}\sum_x |P_1(x) - P_2(x)|\,. \tag{2.16}$$

Total variation distance lies in $[0, 1]$, is symmetric, and satisfies the triangle inequality. For two Bernoulli distributions with parameters p_1 and p_2, it reduces to $D_{\text{TV}} = |p_1 - p_2|$, which gives a direct and interpretable measure of how far a biased sampling procedure deviates from its intended target. This metric appears in the theoretical guarantees developed in Chapter 4, bounding the distance between the output of a verbalized rejection sampler and the true target distribution.

2.3 LANGUAGE MODELS AND IN-CONTEXT LEARNING

Sections 2.1 and 2.2 developed probabilistic tools that operate on numerical representations. This section introduces the second paradigm: computation in natural language. We first describe autoregressive language models as probability distributions over token sequences (Section 2.3.1). We then describe the prompting and in-context learning capabilities that allow these models to solve tasks they were never explicitly trained for (Section 2.3.2).

2.3.1 *Autoregressive Language Modeling*

A language model defines a probability distribution over sequences of tokens drawn from a vocabulary $\mathcal{V}$. Given a sequence $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$, the joint probability factorizes autoregressively via the chain rule of probability:

$$p(\boldsymbol{t}) = p(t_1) \prod_{i=2}^{n} p(t_i \mid t_1, \ldots, t_{i-1}) \,. \tag{2.17}$$

Each conditional $p(t_i \mid t_{<i})$ is a categorical distribution over $\mathcal{V}$, parameterized by a neural network with parameters $\boldsymbol{\theta}$. This factorization is exact, not an approximation: it is simply the chain rule applied to the joint distribution, with no conditional independence assumptions. The factorization in Equation (2.17) is worth pausing on: despite being a neural network trained on raw text, a language model defines a probability distribution in the sense of Section 2.1. This perspective matters for Chapter 4, where sampling from a language model is treated as a stochastic procedure whose bias can be quantified and, in some cases, corrected.

The dominant architecture for modern language models is the transformer [41]. Its central mechanism, self-attention, allows each position in the sequence to attend to all preceding positions, computing context-dependent representations that capture long-range dependencies. The context window, typically spanning thousands to millions of tokens, functions as the model's working memory: everything within the window can influence the prediction at the current position. For this thesis, the architectural details (queries, keys, values, multi-head attention) are not essential; what matters is that transformers are expressive enough to learn complex conditional distributions over token sequences, and that increasing model size and training data systematically improves this ability.

TRAINING OBJECTIVE. A language model is trained by next-token prediction: given a large corpus of text, minimize the cross-entropy loss

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(t_i \mid t_{<i}) \,, \tag{2.18}$$

where the sum runs over all N tokens in the training corpus. This objective connects directly to Section 2.1: averaged over the data distribution, the loss

equals the cross-entropy $H(p_{\text{data}}, p_{\boldsymbol{\theta}})$, which by Equation (2.5) decomposes as $H[p_{\text{data}}] + D_{\text{KL}}(p_{\text{data}} \| p_{\boldsymbol{\theta}})$. Since the entropy of the data is a constant independent of $\boldsymbol{\theta}$, minimizing the cross-entropy loss is equivalent to minimizing the KL divergence from the data distribution to the model. A striking empirical regularity is that this loss decreases as a power law in model size, dataset size, and compute [42], a finding that drove the scaling of language models from millions to hundreds of billions of parameters.

GENERATION AS SAMPLING. At inference time, generating text means sampling from the learned distribution: draw $t_i \sim p_{\boldsymbol{\theta}}(t_i \mid t_{<i})$ sequentially, feeding each sampled token back as context for the next. Each step is a direct draw from a categorical distribution over the vocabulary. Three levers shape the resulting distribution over generated text: the model itself (the parameters $\boldsymbol{\theta}$), the generation procedure (how draws are made from $p_{\boldsymbol{\theta}}$), and the prompt $t_{<i}$ (the conditioning context). Standard autoregressive sampling can produce repetitive or degenerate text, so several modifications to the generation procedure are common. Temperature scaling replaces $p(t)$ with $p_\tau(t) \propto p(t)^{1/\tau}$: higher τ increases randomness, lower τ concentrates mass on the most likely tokens. Top-k sampling restricts the draw to the k most probable tokens [43], and nucleus sampling restricts it to the smallest set whose cumulative probability exceeds a threshold p [44]. The prompt is a complementary lever: changing the conditioning context shapes the resulting distribution without modifying the model or the sampling procedure. The gap between a model's ability to describe a distribution and its ability to sample from it faithfully under a prompt is taken up in Chapter 4.

2.3.2 *Prompting, In-Context Learning, and Chain-of-Thought*

The probability-distribution view of Section 2.3.1 describes how language models are trained. What makes them a new kind of computational object is how they are used at inference time: by conditioning on natural-language prompts, a trained model can be made to solve tasks it was never explicitly trained for, without any update to its parameters.

IN-CONTEXT LEARNING. In-context learning (ICL) [5] refers to the ability of a language model to perform a task given a few input-output examples placed in the prompt. The model conditions on these examples and generates an output for a new input, effectively inferring the underlying task from the demonstration. No gradient updates occur; the model's weights remain frozen, and only the input changes. This can be viewed as a form of implicit Bayesian inference: given the examples (data), the model computes an implicit posterior over tasks and generates outputs from the corresponding predictive distribution [45]. The special case with no examples, only a task description, is called *zero-shot*; with examples, *few-shot*. The range of tasks accessible this way is strikingly broad, spanning translation, arithmetic, and

symbolic reasoning. ICL also becomes substantially more reliable at scale, with larger models solving tasks that smaller ones cannot [12].

THE LLM AS A COMPUTATIONAL MACHINE. This prompting framework invites a structural analogy that Chapter 4 develops in detail. Just as von Neumann's stored-program architecture unifies programs and data in the same memory, an LLM unifies instructions, context, and outputs in the same token space. A text prompt serves as a program that specifies what the model should compute, and the model's response is the output of that computation. This parallel suggests treating natural language not merely as a communication medium but as a genuine medium of computation, where the prompt is the program and the language model is the processor.

CHAIN-OF-THOUGHT REASONING. A further development revealed that the model's *intermediate output*, not just the input, affects the quality of the computation. Wei *et al.* [46] showed that prompting models to produce intermediate reasoning steps dramatically improves performance on arithmetic, commonsense, and symbolic reasoning tasks. Kojima *et al.* [47] showed that CoT reasoning can be triggered zero-shot, without any in-context examples, by simply appending "Let's think step by step" to the prompt. The mechanism is autoregressive: the model's next-token predictions are conditioned on its own previous tokens, so producing intermediate steps changes the conditional distribution over the final answer. The reasoning steps are not merely an explanation or an ornament; they are part of the computation.

LLM-BASED OPTIMIZATION. The capabilities above, ICL and chain-of-thought, operate within a single forward pass (or a single generation). A natural extension is to use language models *iteratively*, feeding outputs back as inputs in a loop. Yang *et al.* [48] showed that LLMs can improve solutions by being shown previous attempts alongside their performance scores. The model acts as an optimizer in language space: it takes a current solution (text), feedback (text describing performance), and produces an updated solution (text). The optimization happens entirely in natural language, with no gradients or numerical parameter updates. This is the direct precursor to the verbalized machine learning framework in Chapter 4, where the object being optimized is a machine learning model parameterized entirely by text.

CAPABILITIES AND LIMITATIONS. In-context learning, chain-of-thought reasoning, and LLM-based optimization demonstrate that language models can perform increasingly sophisticated computational tasks through prompting alone. They suggest that natural language is a surprisingly effective medium for computation. But they also reveal systematic limitations. Small changes in prompt phrasing can drastically affect outputs. Models produce confident but incorrect outputs, a phenomenon known as hallucination. And there is a documented gap between what a language model can describe and what it can faithfully sample from: Gu *et al.* [49] show that LLMs can accurately identify the parameters of a distribution from a small

dataset yet fail to produce samples with matching frequencies, and Van Koevering & Kleinberg [50] document the same asymmetry for coin-flip tasks. The reliability of language-mediated computation under these systematic failures is taken up in Chapter 4.

2.4 FROM PROBABILISTIC TO VERBALIZED REASONING

The probabilistic toolkit of Sections 2.1 and 2.2 and the language models of Section 2.3 share more structure than first appears. The VAE encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ of Section 2.2.1 maps data to latent representations through amortized inference; in-context learning likewise maps prompt examples to an implicit task representation through a forward pass over a frozen network. Both are amortized, both are approximate, and both introduce systematic biases.

This shared structure reflects a deeper pattern. Probability theory has long been the right tool for disciplining unreliable computational media. Shannon's coding theory makes communication reliable over noisy channels; von Neumann's threshold theorem makes computation reliable with faulty components; rejection sampling produces exact samples from intractable targets; variational inference makes intractable posteriors tractable. In each case the medium had known failure modes, and probability theory provided algorithms that compensated for them. Language is an unreliable medium of a different kind, with failures that are semantic rather than physical: ambiguity instead of bit flips, hallucination instead of hardware failure, systematic sampling bias instead of thermal noise. But the structural shape of the problem is the same: a target computation we want to perform reliably, and a medium that introduces errors. The question this thesis takes up is how to make computation in this new medium possible and reliable. Probability theory is a natural starting point: its vocabulary, its algorithms, and its long history of disciplining analogous media all apply.

Yet prior to this thesis, the probabilistic perspective had been applied to computation in natural language only in scattered cases. Core operations such as iterative model refinement, sampling correction, and principled information allocation had not been systematically adapted to the new medium, and no unified treatment of how to make verbalized computing both possible and reliable existed. The next two chapters take up this gap: Chapter 3 strengthens the probabilistic toolkit in numerical settings, and Chapter 4 develops frameworks and tools for verbalized computing.

3

PROBABILISTIC REASONING AND LEARNING

Deep latent variable models such as variational autoencoders (VAEs) route data through an information bottleneck of latent variables (Section 2.2). The tightness of that bottleneck sets the rate, how much information the latent code carries, and shapes what the model is good for: a reconstruction, a sample from the generative distribution, or a compact representation for a downstream classifier. Two design-level questions sit underneath the usual architectural choices. First, when the bottleneck spans multiple latent layers, how should information be distributed across them, and does the best distribution depend on the task? Second, when a trained VAE is handed unseen data, which components actually generalize, and which overfit?

This chapter collects two contributions that answer these questions from complementary angles. Section 3.1 works in the vocabulary of information theory and rate–distortion: it replaces the scalar Lagrange multiplier β of a β-VAE with a per-layer vector $\boldsymbol{\beta}$, giving practitioners independent control over how much information each latent layer carries. A large-scale sweep shows that the best allocation differs across reconstruction, generation, and representation learning, so no one HVAE fits all. An information-theoretic bound on classification accuracy explains the direction in rate space that helps classification. Section 3.2 works in the vocabulary of deep learning and generalization: it decomposes the test-set ELBO into three independent gaps, generalization, amortization, and robustness, showing that the encoder is more susceptible to overfitting than the decoder. Two standard deep learning levers close all three: training on samples from a stronger generative model (DMaaPx, a form of cross-model-class distillation) and adding capacity at the latent dimension axis $\Theta_{\boldsymbol{z}}$. Taken together, the two contributions treat the design of a probabilistic representation, how information is allocated across latents and how resistant each component is to overfitting, as a subject of study in its own right, alongside the inference procedure that operates on it.

3.1 TRADING INFORMATION BETWEEN LATENTS IN HIERARCHICAL VARIATIONAL AUTOENCODERS

Our paper "Trading Information between Latents in Hierarchical Variational Autoencoders" was presented at the International Conference on Learning Representations (ICLR) in 2023 and published in its conference proceedings [13].

Information Allocation in Hierarchical Representations

The variational autoencoder routes data through a bottleneck of latent variables (Section 2.2). The tightness of that bottleneck sets the rate: how much information the latent code carries about the input. A high rate preserves detail; a low rate keeps the latent space close to the prior but loses information the decoder needs. The β-VAE [51] makes this trade-off explicit by placing a scalar Lagrange multiplier β on the rate term. Under the information-bottleneck and rate–distortion reading of Alemi *et al.* [10], Tishby & Zaslavsky [52], and Alemi *et al.* [53], different β correspond to different operating points on a rate–distortion curve, and the practitioner chooses one that matches the downstream task.

Hierarchical VAEs [31–34] extend this picture. The representation now spans multiple latent layers, and each layer $\boldsymbol{z}_\ell$ can carry a different portion of the information in $\boldsymbol{x}$. In principle, information can be redistributed across layers without changing the total rate. Yet practitioners typically tune a single scalar β for the whole model, enforcing the same bottleneck strength at every layer. Different tasks may prefer different allocations, which a single scalar cannot express.

The question is whether the scalar β can be replaced by a per-layer vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_L)$. This requires the total rate to decompose cleanly into per-layer contributions, each controllable independently. Figure 3.1 previews the answer. The three coordinate axes show measured performance on three canonical downstream tasks, reconstruction, generation, and representation learning for classification, for two layer hierarchical VAEs trained on SVHN with many different $\boldsymbol{\beta} = (\beta_1, \beta_2)$. The optimum for each task sits in a different region of rate space, and a conventional β-VAE constrained to $\beta_1 = \beta_2$ (the dashed red lines in the right panels) is optimal for none of them. No one HVAE fits all.

Per-Layer Rates and the Information-Trading Objective

The hierarchical ELBO and the per-layer rate decomposition were previewed in Section 2.2. We briefly restate what is needed, then develop two further ingredients: the class of inference architectures for which the decomposition is clean, and the generalized objective that uses it to trade information between layers.

Consider an HVAE with L layers of latent variables $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$. The generative model factorizes top-down as $p_\theta(\boldsymbol{x}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_L) = p_\theta(\boldsymbol{z}_L) \prod_{\ell=1}^{L-1} p_\theta(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell}) \cdot p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_{\geq 1})$. The inference model has three natural factorizations (Figure 3.2). The *bottom-*

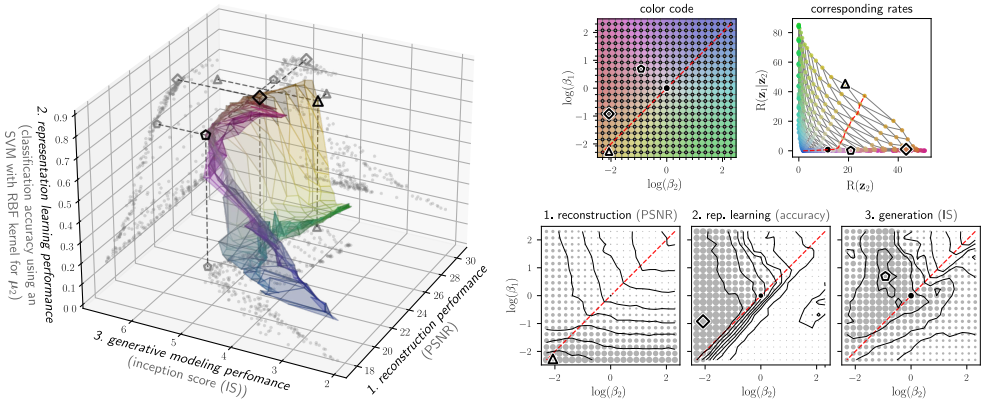


FIGURE 3.1: Trade-off between performance in the three application domains of VAEs, using a two-layer HVAE trained on SVHN. Left: three-dimensional plot of performance, with higher values better along each axis; gray dots on the walls show two dimensional projections. Right: color code, corresponding layer-wise rates, and individual performance landscapes (dot size $\propto$ performance). The per-layer hyperparameters β_2 and β_1 allow tuning the HVAE for best reconstruction ($\triangle$), best generative modeling ($\pentagon$), or best representation learning ($\diamond$). Performance landscapes differ strongly across tasks, and neither a standard VAE ($\beta_2 = \beta_1 = 1$, marked "$\bullet$") nor a conventional β-VAE ($\beta_2 = \beta_1$, dashed red line) is optimal for any of the three applications. Figure adapted from Xiao & Bamler [13].

up model factorizes $q_\phi(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L \mid \boldsymbol{x}) = q_\phi(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{\ell=2}^{L} q_\phi(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{\ell-1})$. The *implicit top-down* model (e.g., Ladder VAE [31]) applies a deterministic feature extractor $\boldsymbol{d}_1 = \boldsymbol{d}_1(\boldsymbol{x})$ and then samples top-down. The *explicit top-down* model generalizes the implicit variant by conditioning each $\boldsymbol{z}_\ell$ directly on the data and the layers above: $q_\phi(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell}, \boldsymbol{x})$.

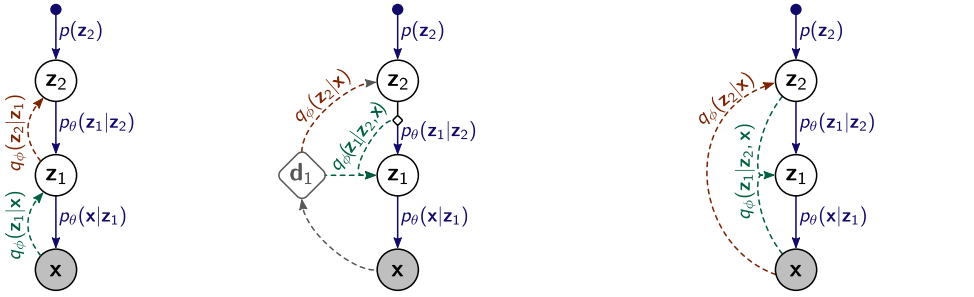
The distinction matters because the total rate decomposes cleanly into per-layer contributions only for the explicit top-down class. Under that factorization, the total rate $R = D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L)\big)$ splits into

$$R = \sum_{\ell=1}^{L} R_\ell, \qquad R_\ell = \mathbb{E}_{q_\phi(\boldsymbol{z}_{>\ell} \mid \boldsymbol{x})}\Big[D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell}, \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell})\big)\Big], \tag{3.1}$$

where R_ℓ is the rate consumed by layer ℓ, averaged over the layers above it. The bottom-up model does not admit this decomposition, because its inference conditionals do not align with the generative conditionals layer by layer. For the remainder, we assume the explicit top-down class.

The information-trading objective replaces the scalar β by a per-layer vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_L)$:

$$\mathcal{L}_{\boldsymbol{\beta}} = D + \sum_{\ell=1}^{L} \beta_\ell R_\ell, \qquad D = -\mathbb{E}_{q_\phi(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L \mid \boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_1, \ldots, \boldsymbol{z}_L)\big]. \tag{3.2}$$



(a) bottom-up (b) implicit top-down (e.g., LVAE) (c) generalized (explicit) top-down

FIGURE 3.2: Inference (dashed) and generative (solid) models for two-layer hierarchical VAEs. (a) Bottom-up inference threads information from the data through each latent in turn. (b) Implicit top-down inference (e.g., LVAE), where $\boldsymbol{d}_1$ is a deterministic function of the data. (c) Generalized (explicit) top-down inference, in which each latent is conditioned directly on both the data and the layers above it. Per-layer rate decomposition and independent control apply to the explicit top-down class in (c), which contains the implicit variant in (b) as a special case. Figure adapted from Xiao & Bamler [13].

Each β_ℓ acts as an independent Lagrange multiplier on layer ℓ's bottleneck. A larger β_ℓ reduces the rate R_ℓ at layer ℓ; tuning the vector $\boldsymbol{\beta}$ therefore reallocates rate across layers. Tying all β_ℓ together recovers the conventional β-VAE, and $\boldsymbol{\beta} = \mathbf{1}$ recovers the standard ELBO. For $\boldsymbol{\beta} \neq \mathbf{1}$, Equation (3.2) is no longer a lower bound on the marginal likelihood, consistent with the rate–distortion reading of Alemi *et al.* [10]: the model trades reconstruction fidelity against per-layer information content, and the practitioner chooses the point that matches the task.

Empirical Findings: No One HVAE Fits All

The framework predicts that different downstream tasks should prefer different allocations of rate across layers. Whether this holds in practice is an empirical question. We sweep $\boldsymbol{\beta}$ over a fine grid and measure performance on three canonical tasks.

SETUP. We train two-layer HVAEs with the explicit top-down inference architecture on SVHN [54], CIFAR-10 [55], and MNIST [56], sweeping $\boldsymbol{\beta} = (\beta_1, \beta_2)$ over a 21×21 log-scale grid from 0.1 to 10 in each coordinate (441 models per dataset). Three metrics are reported: reconstruction quality (PSNR of reconstructed test images), generative sample quality (Inception Score [57] of prior samples), and representation-learning quality (classification accuracy of probe classifiers trained on the latents). The total rate, per-layer rates R_1 and R_2, and distortion D are recorded for each model.

THREE OPTIMA IN THREE DIFFERENT REGIONS. Figure 3.1 shows the three tasks are optimized at different points in rate space. Best reconstruction (marker △) corresponds to a high total rate distributed across both layers, consistent with the rate–distortion trade-off. Best generation (marker ⬠) lies in the region $\beta_2 < \beta_1$, where the top layer z_2 carries more rate than in a comparable β-VAE; decomposing Inception Score into diversity and sharpness shows that sharpness, which depends primarily on $R(z_2)$, drives this preference. Best representation learning (marker ◇) lies in a third region, with a balance between layers that favors downstream classification. The three optima are distinct.

THE CONVENTIONAL β-VAE IS OFF-OPTIMUM EVERYWHERE. Restricting $\boldsymbol{\beta}$ to the diagonal $\beta_1 = \beta_2$ (the dashed red lines in Figure 3.1) gives the conventional β-VAE. The diagonal misses all three task specific optima: in each case, performance is improved by trading rate between z_2 and z_1 rather than changing both bottlenecks together. The standard ELBO point $\boldsymbol{\beta} = \mathbf{1}$ (marked •) is similarly suboptimal. Tying all β_ℓ together, the default in most HVAE codebases, is suboptimal for all three tasks and can be fixed with a one line change to the loss.

THE LANDSCAPE IS SMOOTH. The three performance landscapes in the right panels of Figure 3.1 are smooth in (R_1, R_2), so a coarse sweep is enough to locate a good allocation for a given task. The qualitative shape of the landscapes, and the fact that the three tasks prefer different regions, is consistent across CIFAR-10 and MNIST; the exact location of the optima depends on architecture and dataset.

Information-Theoretic Bounds on Downstream Performance

The empirical landscapes admit a theoretical explanation via information theory. Classification accuracy from a latent is bounded by the mutual information between the latent and the class label. This turns the empirical observation that classification needs the right $\boldsymbol{\beta}$ into a quantitative bound in terms of the per-layer rates.

The bound combines two ingredients. The data-processing inequality (Section 2.1) applied to the Markov path $y \to \boldsymbol{x} \to \boldsymbol{z}_\ell \to \hat{y}$ gives $I_q(y; \boldsymbol{z}_\ell) \leq I_q(\boldsymbol{x}; \boldsymbol{z}_\ell)$. A further application, combined with the non-negativity of KL divergence, bounds $I_q(\boldsymbol{x}; \boldsymbol{z}_\ell)$ by the cumulative rate $R(\boldsymbol{z}_{\geq \ell}) := R(\boldsymbol{z}_L) + R(\boldsymbol{z}_{L-1} \mid \boldsymbol{z}_L) + \cdots + R(\boldsymbol{z}_\ell \mid \boldsymbol{z}_{>\ell})$ [6]. A classical relation between mutual information and classification accuracy [58] gives

$$\text{classification accuracy} \;\leq\; f^{-1}\big(I_q(y; \boldsymbol{z}_\ell)\big) \;\leq\; f^{-1}\big(\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[R(\boldsymbol{z}_{\geq \ell})]\big), \tag{3.3}$$

where f is a monotonically increasing function determined by the label entropy and the number of classes.

Two consequences follow. First, the bound explains why classification performance saturates: once R_ℓ is large enough to saturate $I(y; \boldsymbol{z}_\ell)$, further rate encodes label-irrelevant detail and does not help. Second, the bound indicates which *direction* in rate space helps classification: increasing the cumulative rate $R(\boldsymbol{z}_{\geq \ell})$ up to the layer the classifier operates on raises the upper bound on accuracy, whereas rate allocated

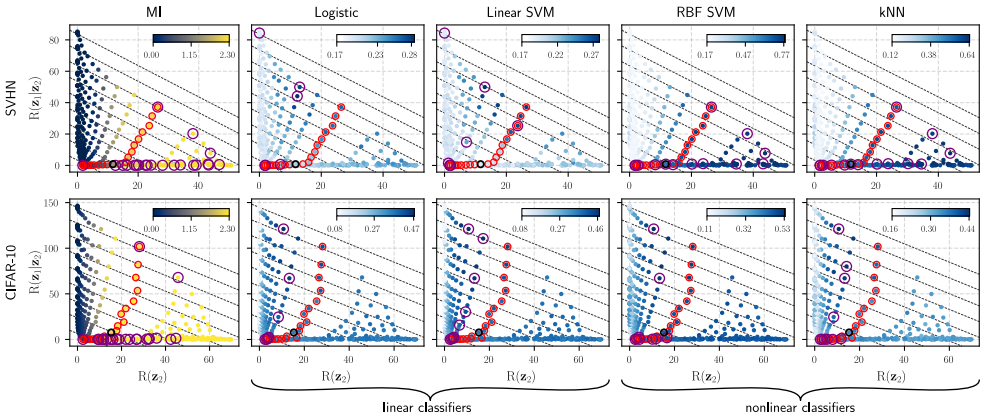


FIGURE 3.3: Mutual information $I_q(y; z_2)$ (first column) and the classification accuracies of four probe classifiers (remaining columns) as a function of the per-layer rates $R(z_2)$ and $R(z_1 \mid z_2)$, for two-layer HVAEs trained on SVHN (top row) and CIFAR-10 (bottom row). Classifiers operate on the encoder mode $\boldsymbol{\mu}_2 := \arg\max_{z_2} q_\phi(z_2 \mid x)$. Black circles "o" mark the standard VAE ($\beta_2 = \beta_1 = 1$), red circles "o" mark conventional β-VAEs ($\beta_2 = \beta_1$), and purple circles "o" mark models that are optimal for a given total rate. Both the mutual information and the measured accuracies peak in the same region of rate space, consistent with the information-theoretic bound in Equation (3.3): when the classifier is expressive enough and the data set is simple (SVHN, top row), higher mutual information corresponds to higher accuracy; for less expressive classifiers or more complex data (CIFAR-10, bottom row), raising $R(z_1 \mid z_2)$ also helps. Figure adapted from Xiao & Bamler [13].

to downstream layers $z_{<\ell}$ does not. Empirically, measured classification accuracy tracks the bound as $\boldsymbol{\beta}$ is swept (Figure 3.3). The bound is loose in absolute terms, as information-theoretic accuracy bounds typically are, but its dependence on $\boldsymbol{\beta}$ matches the direction of empirical improvement.

The bound is modest in scope but useful: it shows that the per-layer rates are information-theoretic quantities controlling downstream performance, and it allows reasoning about allocations without running the full empirical sweep.

Discussion

The paper has two take home messages, one practical and one conceptual. Practically, the default of a single scalar β misses downstream performance that a per-layer $\boldsymbol{\beta}$ can capture; any HVAE with explicit top-down inference already supports per-layer β_ℓ with a one line change to the loss. Conceptually, information allocation across layers is a design axis in its own right. Most prior VAE work has focused on inference, with tighter bounds or more flexible variational families; the proposed framework reallocates information across layers while leaving inference untouched.

The framework has some limitations. First, the explicit top-down class is not universal: popular bottom-up HVAEs do not admit the clean per-layer rate decom-

position and fall outside the framework. Second, the theoretical bound is loose in absolute terms, so it is useful for reasoning about *trends* in rate space rather than for predicting absolute accuracies.

The broader idea is that a representation's usefulness depends not only on how much information it carries but also on where it lives. Section 3.2 takes up the question of when such learned representations generalize.

3.2 GENERALIZATION IN VARIATIONAL AUTOENCODERS: HOW EFFECTIVE IS SYNTHETIC DATA AND OVERPARAMETERIZATION?

Our paper "A Note on Generalization in Variational Autoencoders: How Effective Is Synthetic Data & Overparameterization?" was published in the Transactions on Machine Learning Research (TMLR) in 2024 [14].

Generalization and Overfitting in VAEs

Section 3.1 studied how information is allocated across latents in a fixed VAE architecture. This section asks when the trained encoder and decoder generalize to unseen data. A VAE is trained by maximizing an ELBO on a finite training set, and both components can overfit. Two approximations are being fit at the same time: the decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ is a generative model of the data, and the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is an amortized variational posterior. Either can overfit.

The standard metric for generalization in VAEs is the test-set ELBO. This single number merges at least three failure modes: the generative model overfits, the amortized encoder overfits, and the encoder is fragile to small input perturbations. We therefore decompose it into three separate *gaps*. We then ask how far two standard deep-learning levers, more data and more parameters [38, 39], can close them. Both levers need care in the VAE setting. Naive data augmentation can distort p_{data}, and capacity can be added in more than one place, with different effects. We formalize the three gaps first, then study each lever in turn.

A Three-Gap Diagnostic for Overfitting

We define three gaps that separate encoder from decoder overfitting. Let $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ denote the training and test sets, write $\Theta = (\boldsymbol{\theta}, \boldsymbol{\phi})$ for the decoder and encoder parameters, and let $\text{ELBO}_{\Theta}(\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})]$ be the per-datapoint ELBO. The *generalization gap* is

$$G_g \;=\; \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{train}}}\big[\text{ELBO}_{\Theta}(\boldsymbol{x})\big] \;-\; \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{test}}}\big[\text{ELBO}_{\Theta}(\boldsymbol{x})\big] \;\geq\; 0. \tag{3.4}$$

It measures how much worse the ELBO is on test than on train, but does not attribute the gap to a specific component.

The *amortization gap* [37, 59] isolates the encoder on held-out data. With the decoder frozen, one can further maximize the per-datapoint ELBO over the variational parameters for each test point, giving a stronger variational distribution $q^{\star}(\boldsymbol{z} \mid \boldsymbol{x})$ that does not share weights across inputs. The amortization gap is the ELBO difference between the amortized encoder and this per-point optimum,

$$G_a \;=\; \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{test}}}\big[\text{ELBO}^{\star}_{\boldsymbol{\theta}}(\boldsymbol{x}) - \text{ELBO}_{\Theta}(\boldsymbol{x})\big] \;\geq\; 0, \tag{3.5}$$

where $\text{ELBO}^{\star}_{\boldsymbol{\theta}}$ denotes the ELBO under the optimized per-point $q^{\star}$. A larger G_a means the encoder has overfit to $\mathcal{D}_{\text{train}}$.

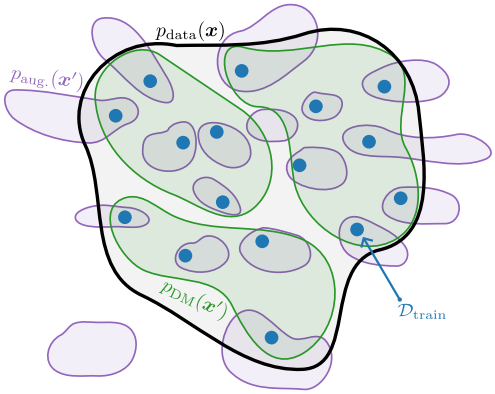


FIGURE 3.4: Illustration of different training distributions. The blue dots are the finite training set $\mathcal{D}_{\text{train}}$, sampled from the unknown data distribution p_{data} (dark-edged region); a diffusion model p_{DM} pre-trained on $\mathcal{D}_{\text{train}}$ (green region) covers a continuous distribution from which the VAE can draw arbitrarily many training samples. The purple region, by contrast, depicts classical data augmentation p_{aug}, which extrapolates from individual points in $\mathcal{D}_{\text{train}}$ and has density outside the support of p_{data} (e.g. when a flipped "2" is no longer a valid digit). Figure adapted from Xiao, Zenn & Bamler [14].

The *robustness gap* measures how much the encoder's output changes under a small input perturbation. For each test point $\boldsymbol{x}^{\text{r}}$, an adversarial perturbation $\boldsymbol{\epsilon}$ with $\|\boldsymbol{\epsilon}\|_\infty \leq \delta$ is found by maximizing the symmetrized KL between $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{\text{r}})$ and $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{\text{r}} + \boldsymbol{\epsilon})$ [60]. Writing $\boldsymbol{x}^{\text{a}} = \boldsymbol{x}^{\text{r}} + \boldsymbol{\epsilon}$ and letting $\tilde{\boldsymbol{x}}^{\text{r}}, \tilde{\boldsymbol{x}}^{\text{a}}$ be the VAE's reconstructions of $\boldsymbol{x}^{\text{r}}$ and $\boldsymbol{x}^{\text{a}}$, the robustness gap is

$$G_r \;=\; \mathbb{E}_{\boldsymbol{x}^{\text{r}} \sim \mathcal{D}_{\text{test}}} \mathbb{E}_{\boldsymbol{x}^{\text{a}} \sim p(\boldsymbol{x}^{\text{a}} \mid \boldsymbol{x}^{\text{r}})} \big[\text{MS-SSIM}(\boldsymbol{x}^{\text{r}}, \boldsymbol{x}^{\text{a}}) - \text{MS-SSIM}(\tilde{\boldsymbol{x}}^{\text{r}}, \tilde{\boldsymbol{x}}^{\text{a}}) \big], \tag{3.6}$$

where MS-SSIM [61] is the multi-scale structural similarity metric (higher is more similar). Because $\boldsymbol{\epsilon}$ is small, MS-SSIM$(\boldsymbol{x}^{\text{r}}, \boldsymbol{x}^{\text{a}})$ is close to one; an overfit encoder produces reconstructions that differ more than the inputs do, so G_r is large.

The three gaps move independently in our experiments: a fix that closes one need not close the others.

More Data: Diffusion Models as a Proxy for p_{data}

The first lever is data. In supervised learning, more training data reduces the generalization gap because the empirical risk approaches the population risk. We ask whether the same holds for all three gaps in a VAE, and where the extra data should come from.

Classical data augmentation perturbs training points with handcrafted transformations to produce $\boldsymbol{x}' \sim p_{\text{aug}}(\boldsymbol{x}' \mid \boldsymbol{x})$. This can help, but p_{aug} only extrapolates from individual training points, relies on assumed invariances that may not hold, and can place density outside the support of p_{data} (Figure 3.4). Another option,

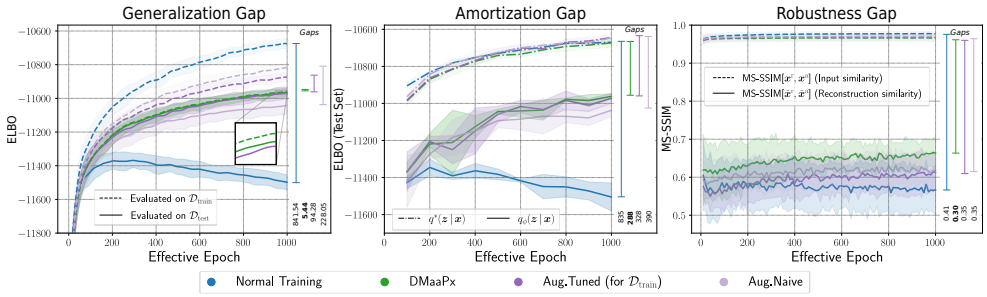


FIGURE 3.5: Three gaps during training on CIFAR-10 [55], for a VAE trained on $\mathcal{D}_{\text{train}}$ alone (normal training) and for a VAE trained on samples from p_{DM} (DMaaPx). The generalization gap G_g (left), amortization gap G_a (middle), and robustness gap G_r (right) are all smaller under DMaaPx, which also achieves the highest test ELBO. The gaps widen over the course of training under normal training, consistent with the encoder progressively memorizing $\mathcal{D}_{\text{train}}$; under DMaaPx, where the VAE sees fresh samples from p_{DM}, they stay small. Figure adapted from Xiao, Zenn & Bamler [14].

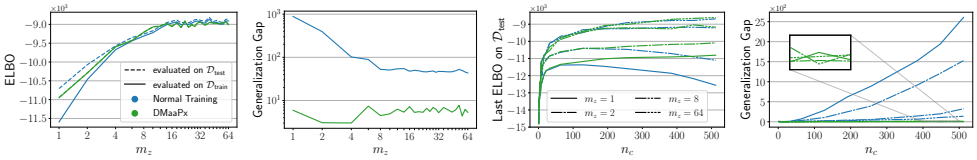
sampling from the VAE's own generative model, is always available but is known to collapse across generations when a generator is trained on its own output [62, 63].

A third option uses a stronger generative model from a different family to approximate the data distribution. In 2024, diffusion models [35, 36, 64] achieved the highest sample quality on natural-image benchmarks. We call this strategy *diffusion model as a proxy for* p_{data}, or DMaaPx. A diffusion model p_{DM} pre-trained on $\mathcal{D}_{\text{train}}$ is used as an approximation of p_{data}, and the VAE is trained by maximizing

$$\mathcal{L} \;=\; \mathbb{E}_{\boldsymbol{x}' \sim p_{\text{DM}}(\boldsymbol{x}')}\big[\text{ELBO}_\Theta(\boldsymbol{x}')\big], \tag{3.7}$$

replacing the empirical training distribution over $\mathcal{D}_{\text{train}}$ with the continuous distribution p_{DM} from which arbitrarily many samples can be drawn. DMaaPx is a *cross-model-class distillation*: the diffusion model and the VAE come from different generative model families. The different family construction avoids the self consuming collapse, and the resulting VAE still provides the fast, controllable, interpretable latent representations that a diffusion model does not.

Figure 3.5 reports the three gaps on CIFAR-10 [55]. All three shrink under DMaaPx, and DMaaPx also reaches a higher ELBO on $\mathcal{D}_{\text{test}}$ than normal training or classical augmentation. The encoder is the main source of overfitting because it sees the same training image every epoch, whereas the decoder is fed samples from the (changing) posterior and is less likely to see the same input twice [37, 59, 65]. Synthetic data from a stronger generator therefore targets the dominant failure mode. In practice, the benefit saturates once p_{DM} is sampled on the order of ten times the training set, so an unlimited synthetic-data budget is not required.



(a) (b) (c) (d)

FIGURE 3.6: Effect of the two capacity axes on test ELBO and on the generalization gap, sweeping each axis while holding the other fixed, for both normal training and DMaaPx. (a, b) $m_{\boldsymbol{z}}$ axis: sweeping $m_{\boldsymbol{z}}$ from 1 to 64 at fixed $n_c = 256$. Test ELBO (a) rises under normal training and plateaus once $m_{\boldsymbol{z}} \gtrsim 16$; the generalization gap (b) shrinks along the sweep and stays smaller under DMaaPx than under normal training throughout. (c, d) n_c axis: sweeping n_c from 1 to 512 at several fixed $m_{\boldsymbol{z}}$. At small $m_{\boldsymbol{z}}$ (1 and 2), increasing n_c produces the classical inverted-U of overfitting: test ELBO (c) first rises then falls, and the generalization gap (d) grows monotonically. Larger $m_{\boldsymbol{z}}$ (8 and 64) tilts the right-hand side of these curves upward, eliminating the drop. Under DMaaPx the generalization gap remains close to zero across n_c. Figure adapted from Xiao, Zenn & Bamler [14].

More Parameters: Where Overparameterization Helps

The second lever is capacity. Classical learning theory predicts that making a model too large causes overfitting, but deep networks often keep generalizing as they get wider [38, 39]. We ask whether VAEs behave the same way and which part of the architecture matters. A VAE has two places to add capacity. The latent dimension $\mathrm{d}_{\boldsymbol{z}} = m_{\boldsymbol{z}} \times 64$ is controlled by a multiplier $m_{\boldsymbol{z}}$ and sets the information bottleneck; the internal widths of the encoder and decoder networks are controlled by a channel count n_c. These form two distinct *complexity axes*. We accordingly split the parameters into $\Theta_{\boldsymbol{z}}$ (the weights of the last encoder layer and first decoder layer, which interact directly with $\boldsymbol{z}$) and $\Theta_{\neg \boldsymbol{z}}$ (the remaining, governed by n_c).

We find that increasing $m_{\boldsymbol{z}}$ at fixed n_c shrinks all three gaps, mirroring the effect of DMaaPx in Figure 3.5. Figure 3.6 then sweeps each axis while holding the other fixed. Along the $m_{\boldsymbol{z}}$ axis, the test ELBO rises until $m_{\boldsymbol{z}} \approx 16$ and then plateaus; the generalization gap shrinks throughout the sweep. Along the n_c axis, behaviour depends on $m_{\boldsymbol{z}}$: at small $m_{\boldsymbol{z}}$ (1 or 2), the test ELBO shows a classical inverted-U of overfitting and the generalization gap grows with n_c; at larger $m_{\boldsymbol{z}}$ (8 or 64), n_c no longer hurts. Each axis in isolation shows a single descent shape; projecting the two onto one aggregate complexity axis produces the classical double descent curve, consistent with the non-deep explanation of double descent [66]. Data and parameters also combine: DMaaPx on top of larger $|\Theta_{\neg \boldsymbol{z}}|$ holds the generalization gap near zero, whereas normal training overfits more as n_c grows. When compute allows, use both; when it does not, prioritize $\Theta_{\boldsymbol{z}}$.

Discussion

Splitting the test ELBO into three gaps reveals that the encoder is more susceptible to overfitting than the decoder, and both DMaaPx and larger $|\Theta_{\boldsymbol{z}}|$ close all three. When compute is tight, prioritize $\Theta_{\boldsymbol{z}}$ over $\Theta_{\neg \boldsymbol{z}}$. More broadly, DMaaPx is an instance of *cross-model-class distillation*: transferring knowledge from a source to a target model designed with different assumptions or structures. Here the source is a diffusion model and the target is a VAE. Training across families also avoids the degradation observed when generators are trained on their own output [62, 63]. The two questions, whether synthetic data helps and where to put extra parameters, are timely: synthetic content is proliferating online, and scaling is the dominant lever in contemporary deep learning.

We note some limitations. The diffusion model that approximates p_{data} is itself trained on $\mathcal{D}_{\text{train}}$, so by the data processing inequality it cannot contain information beyond $\mathcal{D}_{\text{train}}$ except through its own inductive biases; DMaaPx trades a small amount of training set fidelity for continuity, not for new information. And double descent appears here only along an artificially constructed two-axis trajectory; a more detailed investigation of the phenomenon in VAEs is left to future work.

Both sections of this chapter treated the design of a probabilistic representation, i.e., how information is allocated across latents and how resistant each component is to overfitting, as a subject of study in its own right. The next chapter moves from this numerical medium to natural language.

4

VERBALIZED REASONING AND MACHINE LEARNING

The limits of my language mean the limits of my world.

— Ludwig Wittgenstein [67]

Large language models, trained only to predict the next token in a sequence, have become surprisingly general purpose problem solvers. A model given a description of a task in plain English will often produce a correct answer without any task-specific training: it will translate a sentence, summarize a contract, draft a proof, or write a working program, all from a single prompt [5, 12]. The prompt is the only thing that changes between tasks, yet the behavior of the system changes completely. What does it mean to *compute* when instructions, data, and output share one token stream? What does it mean to *learn* when parameters are paragraphs of English rather than vectors of real numbers? And what does it mean to *sample correctly* from a system that can describe a distribution fluently in words but produces biased draws when asked to sample from it? At the start of this PhD, these questions had no clean answers.

This chapter collects three contributions that begin to answer them. Section 4.1 treats a language model with a prompt as a kind of computer: the prompt is a program, written in natural language rather than in formal syntax, and the pretrained language model is a universal compute unit that runs the program and returns an answer. We call this natural-language-based computing paradigm *verbalized computing*. Section 4.2 introduces Verbalized Machine Learning (VML), in which models are parameterized by natural language and trained by iterative exchange between a learner LLM and an optimizer LLM, without any numerical gradient. VML recovers classical regressors (linear, polynomial) and handles small classification tasks, including a realistic medical-image benchmark, while leaving the numerical weights of the LLM untouched. Section 4.3 extends the paradigm from functions to distributions. We ask whether one can parameterize a distribution with a prompt and whether an LLM can act as a faithful sampler from a distribution described in its own prompt. We find a knowledge-sampling gap: the model can identify which distribution a sample came from, yet its own samples from a described distribution are systematically biased. Verbalized Rejection Sampling (VRS) closes part of that gap by adapting the classical rejection sampling (Section 2.2.2) to a language-based algorithm, and a theoretical analysis identifies when this improvement is provable. Taken together, the three contributions argue that classical computing tasks can be lifted into the natural-language medium, with prompts as programs, parameters as sentences, and algorithmic guarantees that can transfer.

4.1 LARGE LANGUAGE MODELS ARE ZERO-SHOT PROBLEM SOLVERS — JUST LIKE MODERN COMPUTERS

Our paper "Large Language Models Are Zero-Shot Problem Solvers—Just Like Modern Computers" was published in the Harvard Data Science Review in 2025 [15]. An earlier version appeared as a blog post shortly after the release of ChatGPT.[1] We wrote it because, from a computer science background, the connection between large language models and classical computers was immediately striking.

Zero-Shot Problem Solving, from Computers to LLMs

Scaling a language model decreases its loss in a predictable way [42, 68, 69]. That is not what made GPT-3 [5] and ChatGPT [70] surprising. What surprised the community is that, at sufficient scale, the same models also began to solve tasks they had never been explicitly trained on: question answering, translation, summarization, even algorithmic problems [5, 71, 72]. We call this *zero-shot problem solving*, following Radford *et al.* [71]: the model has not been trained on the tested task.[2] Smaller models gave no hint of this ability [12], and how it appears is an open question.

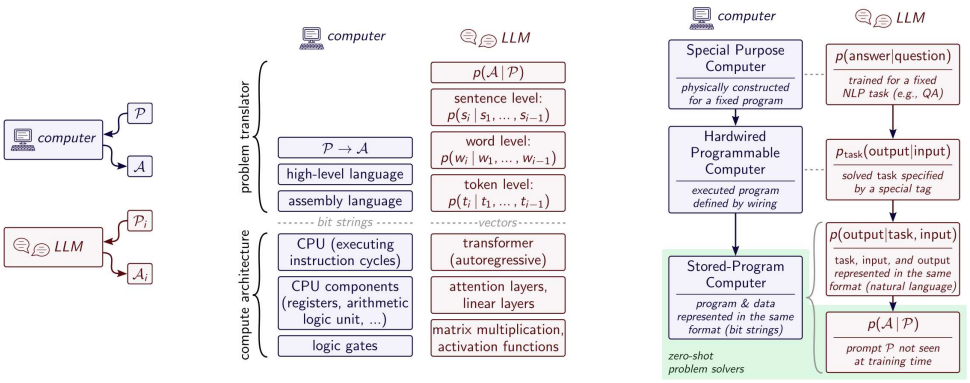
The phenomenon is known as *emergence*: a system showing abilities that cannot be predicted from its smaller versions. Emergence is not specific to language models; physics and biology have discussed it for half a century, with Philip W. Anderson summarising it under the concise title *More Is Different* [73]. It inspired early neural networks such as the Hopfield network [74]. The modern computer shares some similarities. Its elementary components are logic gates implementing simple Boolean operations such as AND and OR; on their own they solve nothing, but wired together at sufficient scale they run everything from calculators to video games. The computer too is a zero-shot problem solver, and the route by which it did so is well understood. The similarities between the two systems imply what LLMs can be used for; the differences teach us how to use them well. Figure 4.1 previews the correspondence along three axes, developed in the next subsection.

A Tale of Two Zero-Shot Problem Solvers

PROBLEM TO ANSWER. Both systems map a problem $\mathcal{P}$ to an answer $\mathcal{A}$. A computer runs a program $\mathcal{P}$ and returns $\mathcal{A}$ deterministically. A language model takes a prompt $\mathcal{P}$ and samples $\mathcal{A}$ from an implicit distribution $p(\mathcal{A} \mid \mathcal{P})$ (Section 2.3.1). The mapping is deterministic in one case and stochastic in the other, but the framework is the same.

1 https://timx.me/blog/2023/computers-vs-llms/

2 Parts of the LLM literature reserve *zero-shot* for prompts with no demonstrations, and *few-shot* for prompts with demonstrations [47]. We treat both as instances of zero-shot problem solving, because neither involves training on the tested task.



(A) Framework (B) Levels of abstraction (C) Paths toward universality

FIGURE 4.1: Three perspectives on the correspondence between a modern computer and a large language model: (a) both produce an answer $\mathcal{A}$ from a problem $\mathcal{P}$, with computers returning $\mathcal{A}$ deterministically and language models sampling $\mathcal{A}$ from a conditional distribution $p(\mathcal{A} \mid \mathcal{P})$; (b) both consist of a problem translator on top of a foundational compute architecture; (c) both evolved from specialized task solvers into general zero-shot problem solvers. Figure copied from Xiao, Liu & Bamler [15].

PROBLEM TRANSLATOR ON TOP OF A COMPUTE ARCHITECTURE. Both systems layer abstraction the same way. A *problem translator* turns a human readable description into the system's native representation (i.e., bit strings or vectors), and a *compute architecture* manipulates that representation. For a computer, the translator is the programming language and its compiler; the architecture is a circuit of logic gates. For a language model, the translator is the natural language and the encoding pipeline; the architecture is a computation graph of matrix multiplications and activation functions [41]. Compute architectures are cheap on their own: a ball rolling on a physical surface minimizes that surface, a perfectly valid computation, but without a translator that turns an arbitrary human-posed problem into such a surface the ball never solves a meaningful problem.

FROM SPECIALIZED TO UNIVERSAL, BY THE SAME STRUCTURAL MOVE. Both systems evolved from specialized task solvers into universal ones. Early computers such as ENIAC [75] were built from arithmetic units and switches; the program lived in the hardware wiring, and each new task required physical rewiring. The von Neumann architecture [76] represented the program as data, in the same memory as the inputs, so the computer could read and execute arbitrary programs. Language models followed an analogous path. Early approaches trained a separate model for each task. Multitask architectures then selected a task specific head by a discrete task tag [77, 78]. The next step was to represent the task as a natural language sentence in the same format as the input [79], and Radford *et al.* [71] trained a pure language model, GPT-2, that solved a range of NLP tasks by having them stated in

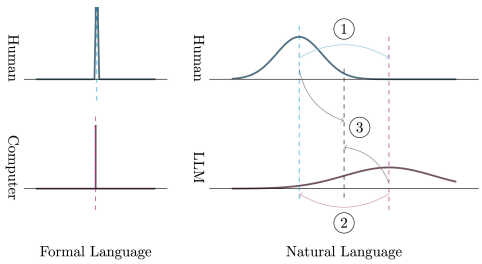


FIGURE 4.2: Communication gaps between a user and a problem solver, for formal and natural language. With a formal language (left), both the user's and the computer's distributions over meanings are close to delta functions and closely aligned, so instructions are unambiguous. With a natural language (right), both distributions are broad (ambiguity) and may be centred at different points (misalignment). Prompting research can be read as a spectrum of methods for closing this gap, ranging from the user adapting toward the model to the model adapting toward the user, with most work falling in between. Figure copied from Xiao, Liu & Bamler [15].

the prompt. In both cases, a single fixed machine (model) can now solve arbitrary tasks stated as part of its input, i.e., from specialized to universal.

PROMPT AS PROGRAM. This gives a concrete reading of what a language model does. The prompt acts as a program, written in natural language rather than in formal syntax. The language model is a fixed compute unit that reads this program and returns an answer. This reading is what the rest of the chapter builds on.

Where the Parallel Breaks

The parallel is useful, but not identical. Formal programming languages are designed to be precise: code has one correct interpretation, and a good programmer is expected to know the language well. The user's and the computer's models of the language are both sharp and aligned. Natural language is neither. A sentence often has many correct interpretations, and both humans and LLMs learn it from data rather than from a specification, so their two models are typically somewhat misaligned (Figure 4.2).

Much of prompting research can be understood as closing this gap. At one end, humans adapt toward the model: signaling phrases like "Let's think step by step" [47] empirically improve answer quality. At the other end, the model is adapted toward the user, through instruction tuning and reinforcement learning from human feedback [80]. Most prompting research sits between the two [81].

Discussion

The reading developed in this section, prompt as program and language model as compute unit, points beyond itself. Classical computing is computation in a symbolic, formal language. With a language model in the role of a compute unit, computation can now happen in natural language itself. We call this natural-language-based computing paradigm *verbalized computing*. The remaining sections of this chapter take the first concrete steps in building it. Section 4.2 shows that classical machine learning carries over, with models parameterized by prompts and trained by iterative rewriting. Section 4.3 shows the same for sampling, embedding classical rejection sampling in a prompt and preserving part of its guarantee.

This paradigm is worth pursuing for reasons deeper than the fact that it is newly feasible. Natural language is what humans already use to think and reason. Computing in natural language thus makes such powerful problem solving machines accessible to the general public, who have not trained in programming. Moreover, Turing envisioned a computing machine that would think and solve problems in natural language, the way humans do [2]. Verbalized computing is thus not only a technical opportunity of the LLM era but the realization of a vision the field has had since its beginnings. Chapter 5 will return to the open questions this raises.

4.2 VERBALIZED MACHINE LEARNING: REVISITING MACHINE LEARNING WITH LANGUAGE MODELS

Our paper "Verbalized Machine Learning: Revisiting Machine Learning with Language Models" was published in the Transactions on Machine Learning Research (TMLR) in 2025 [16].[3]

Learning When Parameters Are Sentences

For most of computing history, programs were written by hand. Machine learning changed this: instead of writing programs, we learn them from data. Section 4.1 argued that the language model with a prompt plays the same structural roles (compute unit and program) in a natural-language-based computing paradigm, and the same historical transition becomes visible in this new medium: once the prompt is a program, it can be written by hand, or it can be learned from data. The language model already does a limited version of the latter through in-context learning (ICL; Section 2.3.2), where a handful of input-output examples in the prompt let the model adapt to an implied task in a single forward pass, which from the perspective of machine learning resembles a nonparametric model. What we describe here is the parametric alternative, making use of a multi-step iterative learning procedure, in which the demonstration data are summarized with natural language into explicit patterns and knowledge that act as the model's parameters.

In machine learning, a model is usually a function approximator whose parameters are real-valued numbers, fit to data by gradient descent. The conceptual move of verbalized machine learning (VML) is to change the parameter space: numerical weights become a natural language prompt, and the pretrained LLM itself, with its weights frozen, can evaluate the function (Figure 4.3). Training becomes an iterative rewrite of that prompt rather than an update of the LLM's weights. Three features fall out of this setup and shape the rest of this section. First, *inductive bias via a prior prompt*: a natural language prefix can encode prior knowledge about the task in the same medium in which the rest of the model speaks. Second, *automatic model selection*: because the parameter space is natural language, a change of model class is just a rewrite of the prompt, so when the current learner fits the data poorly the optimizer can propose a more expressive class (for example, a quadratic in place of a linear regressor). Third, *interpretable training*: the parameters are natural language pattern descriptions, and each update step produces a human readable change together with its own stated reason for making it. In short, VML brings classical

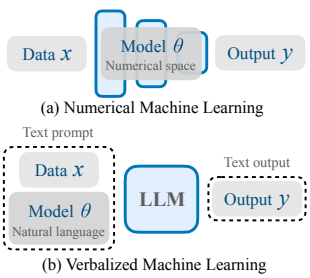


FIGURE 4.3: Numerical machine learning versus VML. Figure copied from Xiao *et al.* [16].

3 An earlier version was released as an arXiv preprint on 6 June 2024.

machine learning into the verbalized computing paradigm of Section 4.1. Once the prompt is a program, learning that program from data is the natural next step.

The Verbalized Learning Loop

Formally, the learner in VML can be denoted as $f(\boldsymbol{x}; \boldsymbol{\theta})$ whose parameters $\boldsymbol{\theta}$ are a sequence of text tokens in Θ_{language}, i.e., the space of natural language. The input $\boldsymbol{x}$ is similarly a token sequence, or an image if the LLM accepts visual input. In a supervised learning setting, given training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$, the natural objective is a supervised loss (e.g., least squares) constrained to $\boldsymbol{\theta} \in \Theta_{\text{language}}$, but the gradient step that would normally solve it is unavailable: back propagation through discrete tokens is sample inefficient, and nothing in a raw gradient step guarantees the update stays inside Θ_{language}.

Because $\boldsymbol{\theta}$ is a text prompt, updating it is a form of prompt optimization: what distinguishes VML from prior works on generic prompt optimization is that the update is driven by the learner's predictions on data, not by a free-form search over candidate strings. Rather than compute a numerical gradient, VML approximates the next step parameters directly, as the optimizer f_{opt} can be specified in natural language in the same spirit of the learner f. Formally, it is a function $f_{\text{opt}}(\boldsymbol{x}, \hat{y}, y, \boldsymbol{\theta}; \boldsymbol{\psi})$ that takes a batch of $(\boldsymbol{x}, \hat{y}, y)$ triples together with the current parameters $\boldsymbol{\theta}_i$ and emits $\boldsymbol{\theta}_{i+1}$. The optimizer's own "parameters" $\boldsymbol{\psi}$ are themselves natural language as well: the optimizer makes an API call to a frozen LLM, and $\boldsymbol{\psi}$ is a prompt that sets its role and update behaviour, for example by specifying the update strength or by carrying a short history of previous steps (like momentum). With these pieces in place, training is the two-step loop in Algorithm 1: the learner predicts across a batch, the optimizer reads the predictions and the batch together with the current prompt and emits a revised prompt, and the cycle repeats for T iterations. No LLM is ever re-trained; only the prompt $\boldsymbol{\theta}$ changes. Figure 4.4 shows the prompt templates for both the learner f and the optimizer f_{opt}.

Algorithm 1:
VML training loop.

Input: initial parameters $\boldsymbol{\theta}_0$; iterations T; batch size M; optimizer prompt $\boldsymbol{\psi}$.
for $i = 1, \ldots, T$ **do**
  sample M training examples $\{(\boldsymbol{x}_m, y_m)\}_{m=1}^{M}$;
  for $m = 1, \ldots, M$ **do**
    $\hat{y}_m \leftarrow f(\boldsymbol{x}_m; \boldsymbol{\theta}_{i-1})$;
  $\boldsymbol{\theta}_i \leftarrow$ $f_{\text{opt}}(\{(\boldsymbol{x}_m, \hat{y}_m, y_m)\}_{m=1}^{M}, \boldsymbol{\theta}_{i-1}; \boldsymbol{\psi})$;
Output: $\boldsymbol{\theta}_T$.

Applications and Case Studies

Unless stated otherwise, the experiments use the instruction tuned Llama-3 70B [82] as both the learner and the optimizer, with $N = 100$ training points, batch size $M = 10$, and Algorithm 1 as the training loop. Deviations from this default are flagged per experiment.

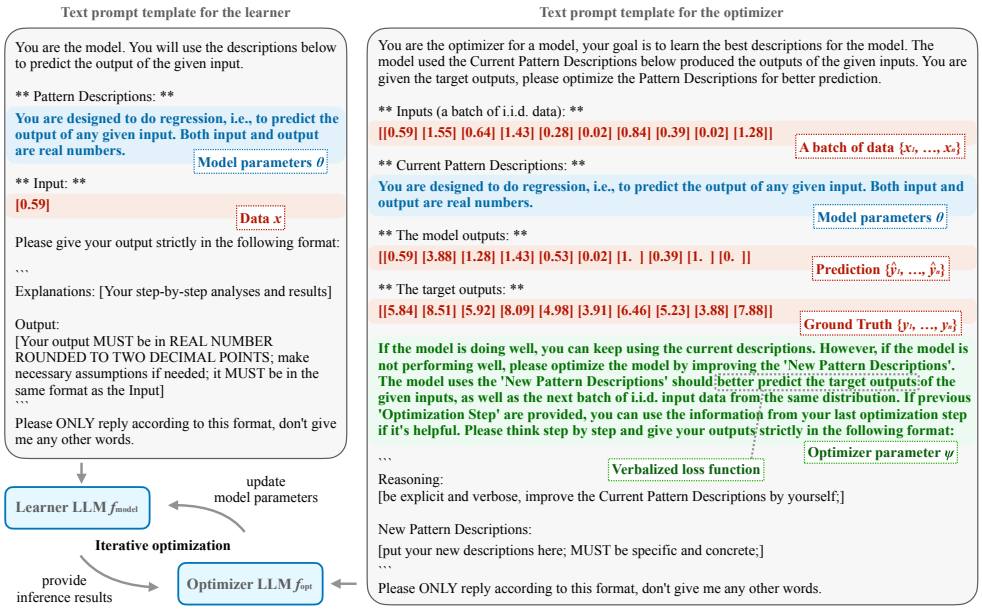
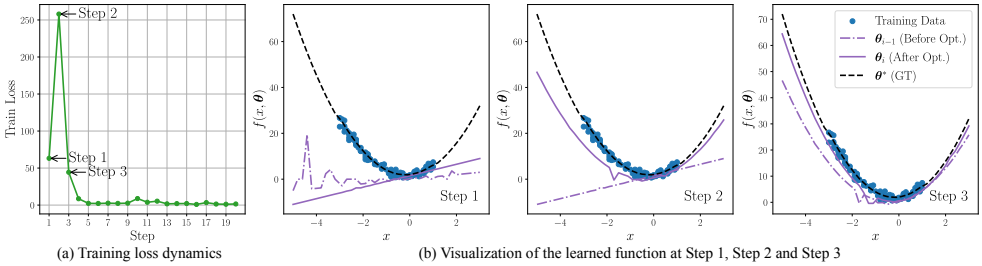


FIGURE 4.4: Prompt templates and the iterative loop in the regression example. The learner prompt (left) describes the regression task and contains the current parameter $\boldsymbol{\theta}$; the optimizer prompt (right) contains a batch of inputs, predictions, and targets together with $\boldsymbol{\theta}$, and asks for an updated $\boldsymbol{\theta}$. Figure copied from Xiao *et al.* [16].

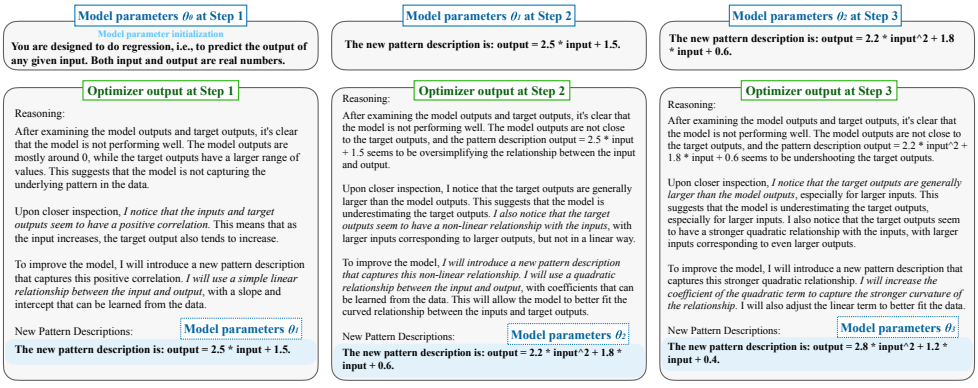
REGRESSION. We first try out a linear regression setup, $y = 3x + 4 + \epsilon$, in which the optimizer settles on a linear model at step 1 and then converges cleanly onto the ground truth coefficients over the next fifteen steps; a more interesting example is the polynomial case (see Figure 4.5), which exercises the same loop together with an additional model-class selection step. The ground truth is $y = 3x^2 + x + 2 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$ and $x \sim \mathcal{U}(-3, 1)$. The initial parameter $\boldsymbol{\theta}_0$ states only that the task is regression from $\mathbb{R}$ to $\mathbb{R}$. At step 1 the optimizer notices that y has a wider range than x with a positive correlation and proposes a linear model; this is the wrong class, and the training loss jumps. At step 2 the optimizer notes the poor fit, recognizes the nonlinearity in the data, and switches to a quadratic. The loss drops sharply. From step 3 onward the optimizer refits coefficients of the quadratic model. The experiment illustrates automatic model class selection together with interpretable training: the optimizer's reasoning for switching from linear to quadratic is visible in its output, and each subsequent refit carries its own visible reasoning.

MEDICAL IMAGE CLASSIFICATION. We move from synthetic regression to a realistic task. The dataset is a 100/100, train/test subset of PneumoniaMNIST [83]: binary classification of chest X-ray images as pneumonia or normal, with the two classes balanced in both splits. The backbone is GPT-4o [84], which accepts visual inputs; training runs for five epochs. We run two configurations. In one, $\boldsymbol{\theta}_0$



(a) Training loss dynamics

(b) Visualization of the learned function at Step 1, Step 2 and Step 3



(c) Model parameters and optimizer output at Step 1, Step 2 and Step 3

FIGURE 4.5: Training dynamics for VML-based polynomial regression, over two epochs of ten steps each. The large loss spike at step 1 reflects the optimizer's initial choice of a linear model, which it revises to a quadratic at step 2. The training loss is not fed into the VML loop; it is logged on the side for monitoring only. Figure copied from Xiao *et al.* [16].

introduces the task as binary image classification. In the other, $\boldsymbol{\theta}_0$ adds a sentence as a prior saying that the input is an X-ray image used to identify pneumonia. Both configurations improve over training, and the with-prior configuration converges faster and reaches a higher test accuracy. Inspecting the learned parameters at step 50, the with-prior learner carries explicit medical vocabulary (for example, "acute infection", "pneumatocele formation"), while the without-prior learner carries generic visual vocabulary ("visible opacities", "uniform texture"; Figure 4.6). The experiment illustrates inductive bias injection together with interpretable training: what the model has learned can be read off the parameter text and be inspected manually, and with and without the prior the two learners read very differently.

VML VERSUS IN-CONTEXT LEARNING. In the perspective of machine learning, ICL can be read as a *nonparametric* method, whereas VML is a *parametric* method, in which the training examples are summarized into explicit patterns rather than carried as raw demonstrations in every query. We test this quantitatively by comparing VML to ICL on linear and polynomial regression, on the medical image task above, and on two small 2D classification tasks (linearly separable two blobs, and non-linearly separable two circles). For ICL we use the best result across five runs;

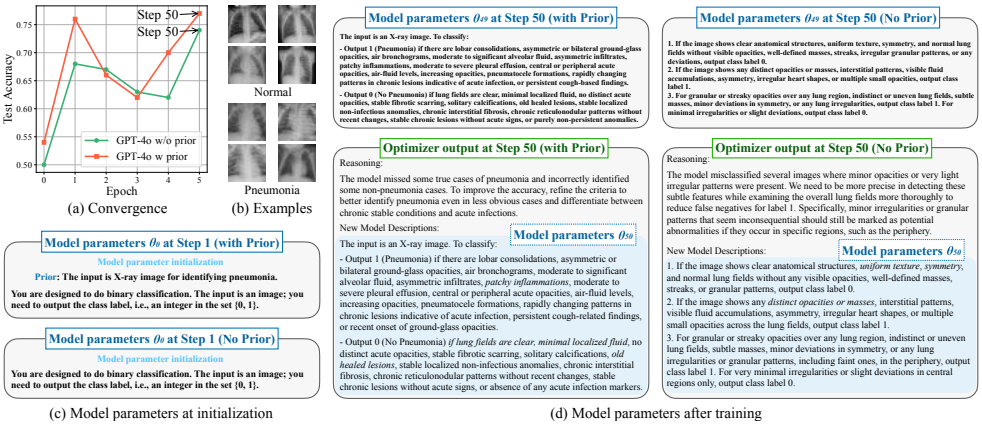


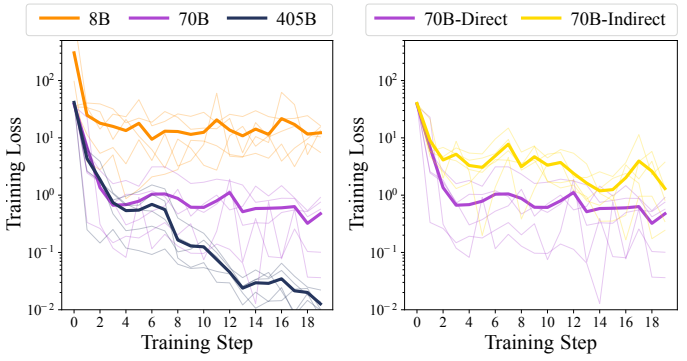
FIGURE 4.6: Tiny-PneumoniaMNIST image classification with VML, for models with and without a pneumonia specific prior at initialization. Figure copied from Xiao *et al.* [16].

VML is trained under the default setup with no prior prompt. VML outperforms ICL substantially on both regression tasks (roughly three-fold on linear regression, and well over an order of magnitude on polynomial regression), ties ICL on the two 2D classification tasks, and improves medical image test accuracy from roughly 48% to 74%.

EFFECT OF SCALING AND OPTIMIZER PARAMETERIZATION. With a stronger backbone, VML should learn faster and reach a lower loss: the optimizer's job is to reason about data and update the parameter prompt accordingly, and that reasoning should improve as the backbone's reasoning ability improves. We test this on the linear regression setup above with Llama-3.1 at three sizes: 8B, 70B, and 405B. Larger backbones do converge faster and reach lower loss (Figure 4.7a). VML's effectiveness scales with the backbone's capability, not against it. The paper also reports an ablation on optimizer parameterizations (Figure 4.7b). The *direct* parameterization is the one used throughout the section: a single LLM call couples the gradient and the update together, $\boldsymbol{\theta}_i = f_{\text{opt}}(\{(\boldsymbol{x}_m, \hat{y}_m, y_m)\}_{m=1}^{M}, \boldsymbol{\theta}_{i-1}; \psi)$. The *indirect* parameterization splits this into two LLM calls, a textual-gradient step that writes down how $\boldsymbol{\theta}$ should change, $\partial\ell/\partial\boldsymbol{\theta} = f_{\text{grad}}(\ell, \hat{y}, \boldsymbol{\theta}_{i-1})$, followed by an update step that applies that change, $\boldsymbol{\theta}_i = f_{\text{update}}(\boldsymbol{\theta}_{i-1}, \partial\ell/\partial\boldsymbol{\theta})$ [85, 86]. The direct version converges faster and with less variance. The paper also includes a qualitative comparison with generic prompt optimizers such as APE [87].

Discussion

Stepping back, VML is a concrete example of how a classical machine learning algorithm can be adapted to run inside the verbalized computing paradigm of Section 4.1. The adaptation is structural: data and parameters now share one



(a) Powerful LLMs improve VML (b) Direct vs. Indirect parameterization

FIGURE 4.7: Training loss on linear regression. (a) Llama-3.1 at three sizes (8B, 70B, 405B): larger backbones converge faster and reach lower loss. (b) Direct versus indirect optimizer parameterization on Llama-3.1-70B: the direct version (a single LLM call per step) converges faster and with less variance than the indirect version (a textual-gradient call followed by an update call). Five individual runs are shown as thin lines, with their mean as a thick line. Figure adapted from Xiao *et al.* [16].

medium (both are token sequences in the prompt), echoing the von Neumann architecture in which instructions and data share the same representation and memory. It is also interpretive, because the parameter and the update step are both in readable language, interpretability is built in rather than studied post-hoc. The broader lesson is a chapter-level one. LLMs provide the engine for computation in natural language, and the classical algorithms can still be effective and can be used to harness the power of such advanced models. Being able to do learning in natural language also makes the powerful tool of machine learning more accessible to the general public. Section 4.3 extends the same verbalized treatment from functions to distributions, asking whether an LLM's sampling behaviour can be disciplined in the same way.

4.3 FLIPPING AGAINST ALL ODDS: REDUCING LLM COIN FLIP BIAS VIA VERBALIZED REJECTION SAMPLING

Our paper "Flipping Against All Odds: Reducing LLM Coin Flip Bias via Verbalized Rejection Sampling" is available as an arXiv preprint [17].

From Function Approximation to Distribution Approximation

Section 4.2 showed that, once a prompt is read as a program, the prompt itself can be treated as a parameter: a natural language sequence that makes a pre-trained LLM act as a function approximator. The natural next question is the analogous one for distributions. Can a probability distribution be parameterized by a natural language prompt, and can the language model act as a faithful sampler from the distribution it describes?

Formally, a distribution parameterized by a prompt is denoted as $P(x;\theta)$, where θ captures both an underlying numerical parameter (for Bernoulli, a single $p \in [0,1]$) and the linguistic phrasing around it. For the same p, different phrasings can induce different sampling behaviours. We treat the language model as a *black-box sampler*: prompt in, text out, no access to internal weights, activations, or token logits. This is both a methodological stance and a design feature. It lets the same method run across opensource and proprietary models identically (we test on Llama-3.1 [88], GPT-4.1-nano [84], DeepSeek-V3 [89], and Qwen-2.5 [90]), and it stays compatible with chain-of-thought reasoning [46], whose intermediate tokens would confound any logit-based approach. Bernoulli is our testbed: the simplest non-trivial case, and one that the recent literature treats as a foundational stress test for LLM sampling [49, 50, 91–93].

When we start to test this picture, we find a *knowledge-sampling gap*. Language models can evaluate which distribution a dataset came from, yet when asked to draw their own samples from a described distribution, the resulting samples are systematically biased (Figure 4.8). The rest of this section first documents this gap under prompt and reasoning interventions, and then closes part of it by transplanting a classical probabilistic algorithm, rejection sampling (Section 2.2.2), into the language interface.

Direct Sampling and Its Limits

Before turning to an algorithmic fix, we document the gap under two kinds of direct intervention on the prompt itself: changing the phrasing of the distribution description, and varying the length of chain-of-thought reasoning.

CAN PROMPT PHRASING REDUCE THE BIAS? A natural first fix is to change the phrasing of the distribution. A Bernoulli prompt can emphasize the probability of generating a 1 (call this phrasing P_1), the probability of generating a 0 (P_0), or both

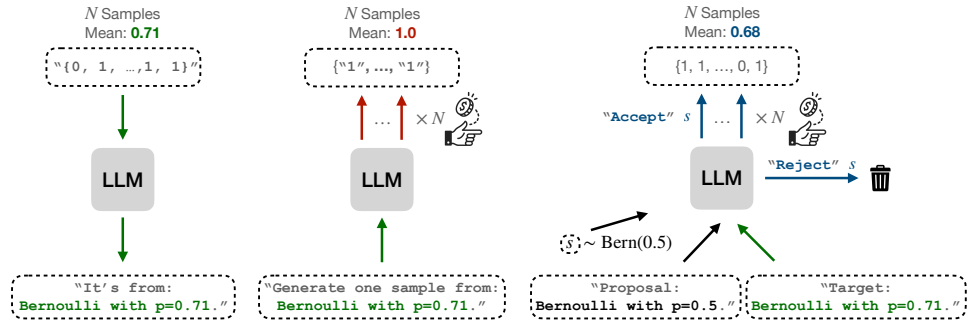


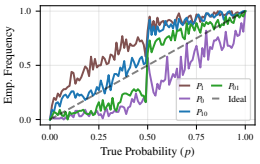
FIGURE 4.8: An overview of the settings we compare. (a) The LLM can identify the Bernoulli distribution that a dataset came from. (b) Asked to draw samples from a described Bernoulli distribution, its empirical frequencies are biased. (c) Verbalized Rejection Sampling (VRS) asks the model to accept or reject each proposed sample, and collects only the accepted ones. Figure copied from Xiao *et al.* [17].

(P_{10} and P_{01}; see Figure 4.9b). To quantify the effect of each phrasing, we introduce the *Sum of TV Distances* (STVD):

$$\text{STVD} = \sum_{i=0}^{100} |\tilde{p}_i - p_i|, \tag{4.1}$$

where $\tilde{p}_i$ is the empirical frequency of 1s obtained from 100 LLM queries at target p_i, summed across the 101 equally spaced values $p_i \in \{0.0, 0.01, \ldots, 1.0\}$. Each term $|\tilde{p}_i - p_i|$ is the total variation distance between the empirical and target Bernoulli distributions, so STVD is a single scalar summary of the whole calibration curve, with lower values better. Balanced phrasings (P_{10}, P_{01}) calibrate more closely than one-sided ones (Figure 4.9a), but no phrasing eliminates the bias. On Llama-3.1 the best phrasing P_{10} reaches an STVD of 12.50, roughly half of the baseline P_1 at 25.36. This extends across models (Table 4.1, 'Direct'). A similar human study shows that asking a person to imagine a coin flip by phrasing that contains only "heads" produces a comparable bias toward heads, and mentioning both sides reduces it only partially [94].

DOES CHAIN-OF-THOUGHT REASONING HELP SAMPLING? A second natural fix is to let the model reason before sampling. We vary the instructed reasoning length from 0 to 500 words, with an additional "Auto" setting that imposes no length constraint, and track STVD as a function of reasoning length. On Llama-3.1, longer chain-of-thought slightly improves calibration. On GPT-4.1-nano and Qwen-2.5 the effect is neither consistent nor monotonic, and in some configurations longer reasoning increases STVD. Therefore, unlike in question answering, chain-of-thought does not reliably improve i.i.d. sampling, and its effect is model-dependent.



(a) Calibration Plot

$P_1(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 1 is p."

$P_0(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 0 is $1-p$."

$P_{10}(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 1 is p, and the probability of 0 is $1-p$."

$P_{01}(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 0 is $1-p$, and the probability of 1 is p."

(b) Four different phrasings for a Bernoulli with p

FIGURE 4.9: Calibration curves for four semantically equivalent phrasings of a Bernoulli, on Llama-3.1. Balanced phrasings (P_{10}, P_{01}) track the ideal diagonal more closely than one-sided ones (P_1, P_0), but all remain visibly biased. Figure copied from Xiao *et al.* [17].

The "Auto" setting is competitive with any fixed length on average, so we use it for the remaining experiments.

Prompt phrasing and reasoning length are two of the main tools the prompt engineering literature has for shaping LLM output, and neither eliminates the sampling bias. If language only interventions are insufficient, the next option is an algorithmic one that embraces the bias and corrects it.

Verbalized Rejection Sampling

Classical probabilistic methods have a direct recipe for turning a biased sample source into an unbiased one, rejection sampling, provided we can evaluate both the target and a simpler proposal distribution. We recap the recipe, adapt it to run inside a natural language prompt.

REJECTION SAMPLING. Rejection sampling (Section 2.2.2) draws samples from a target distribution P using access to a simpler *proposal* distribution Q, given a bound $M < \infty$ on the likelihood ratio $P(x)/Q(x)$. Drawing $x \sim Q$ and accepting it with probability $A(x) = P(x)/(MQ(x))$ returns a sample from P; the overall acceptance rate is $\alpha = 1/M$. Each accept-or-reject decision is itself a Bernoulli trial with parameter $A(x)$, and in the Bernoulli case $M = \max\{p/q, (1-p)/(1-q)\}$.

FROM ALGORITHM TO PROMPT. Verbalized Rejection Sampling (VRS) encodes the three inputs of the classical recipe, the target P, the proposal Q, and a proposed sample $x \sim Q$, as natural language slots in a fixed prompt template (Figure 4.10, right). The language model is instructed to reason about whether to accept the proposed sample, then emit a single letter from $\{T, F\}$. If the response is T, we keep x; otherwise we draw a fresh $x \sim Q$ and repeat until the desired number of accepted samples has been collected. Neither the evaluation of P and Q nor the accept-or-reject step ever leaves the language interface: no logits are read, and no numerical hyperparameters are tuned beyond the default decoding settings.

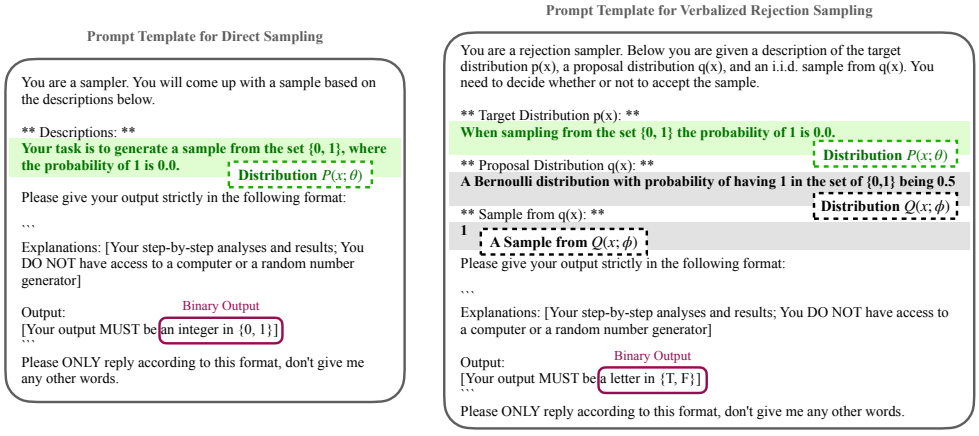
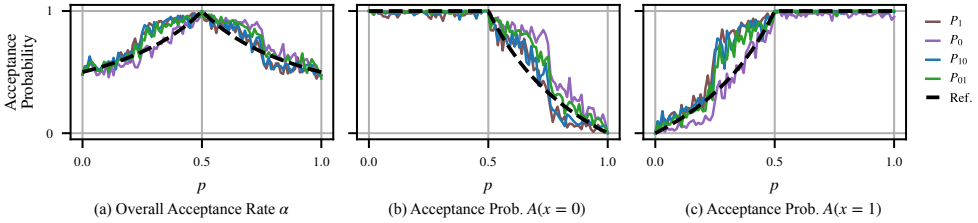


FIGURE 4.10: Prompt templates for direct sampling (left) and VRS (right). The direct template describes the target distribution and asks the model to produce a sample. The VRS template describes the target P, the proposal Q, and a proposed sample $x \sim Q$, and asks the model to accept or reject it. Figure copied from Xiao *et al.* [17].



(a) Overall Acceptance Rate α (b) Acceptance Prob. $A(x = 0)$ (c) Acceptance Prob. $A(x = 1)$

FIGURE 4.12: Empirical acceptance rates for VRS on Llama-3.1, compared with the analytic $A(x)$. The alignment is close, and the remaining error is concentrated on the non-trivial outcome. Figure copied from Xiao *et al.* [17].

EMPIRICAL RESULT. We evaluate VRS on the four language models listed above, with Q fixed to Bern(0.5) using a random number generator and 101 equally spaced target values of $p \in [0, 1]$, and we run VRS until 100 samples have been accepted at each p. Calibration curves under VRS track the ideal diagonal closely (Figure 4.11), and empirical acceptance rates align well with the analytic $A(x)$ (Figure 4.12). The STVD improvement is substantial and consistent across models (Table 4.1, 'VRS'). Most entries see a reduction of more than 50%, and some are closer to 75%. The improvement holds across all four models, indicating the method is model-agnostic.

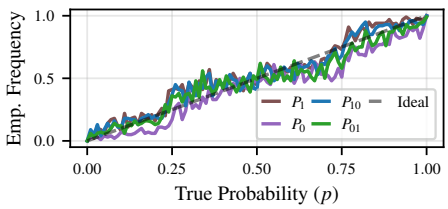


FIGURE 4.11: Calibration curves under VRS on Llama-3.1, for the four phrasings of Figure 4.9. All four track the ideal diagonal. Figure copied from Xiao *et al.* [17].

Method	Llama-3.1 70B					GPT-4.1-nano					DeepSeek-V3					Qwen-2.5 72B				
	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean
Direct	25.36	24.79	12.50	16.59	19.81	17.87	30.23	16.63	19.24	21.00	17.76	19.39	20.78	23.26	20.30	20.73	18.72	19.00	22.64	20.27
VRS	5.73	7.64	5.36	5.60	**6.08**	12.96	13.06	9.50	8.46	**11.00**	5.34	9.06	5.29	6.94	**6.66**	5.93	6.35	4.49	5.12	**5.47**

TABLE 4.1: Sum of TV Distances (STVD, lower is better) for direct sampling and VRS, across four phrasings of the Bernoulli distribution and four language models. The *mean* column is the average over the four phrasings for that model. Table adapted from Xiao *et al.* [17].

Why Does VRS Work?

The empirical result is surprising: VRS delegates the crucial accept-or-reject step, *which is also Bernoulli sampling*, to the same language model that produced biased samples in the first place. If the LLM is a biased Bernoulli sampler, why does wrapping the decision in rejection sampling produce better calibrated samples than direct sampling? We study prompt-level ablations (stripping the external proposal, instructing the model to compute M) and find that the VRS prompt alone cannot recover the full gain, so the explanation must come from the rejection-sampling algorithm itself.

A TOLERATED-BIAS BOUND. We model the LLM as a biased Bernoulli sampler: the accept-or-reject decision is drawn from $\tilde{A}(x) = A(x) + e(x)$ rather than from $A(x)$, with $|e(x)| \le c$ for some bias budget $c \ge 0$. Let $\tilde{P}$ denote the distribution of samples accepted under this biased rule, and let $\bar{P}$ denote the distribution produced by a direct sampler with the same additive bias $\bar{p} = p + e$, $|e| \le c$. Proofs of both results that follow are in the paper's appendix.

Proposition 1 (TV bound). Let $\hat{x}$ denote the non-trivial outcome with $A(\hat{x}) < 1$, and x^* the trivial outcome with $A(x^*) = 1$. If the model is unbiased on the trivial case and $|e(\hat{x})| \le c$, then

$$\mathrm{TV}(\tilde{P}, P) \;\le\; \frac{Q(\hat{x})\, M\, c}{1 - Q(\hat{x})\, M\, c}. \tag{4.2}$$

At $c = 0$ we recover classical rejection sampling with $\mathrm{TV}(\tilde{P}, P) = 0$. The empirical observation that the model is essentially unbiased on $A(x^*) = 1$ (Figure 4.12) makes this the operative bound.

Corollary 1 (when VRS beats direct sampling). Under the assumptions of Proposition 1, VRS has smaller TV to P than direct sampling exactly when

$$\frac{Q(\hat{x})}{\alpha}\,(1 + c) \;<\; 1, \qquad \text{equivalently} \quad c \;<\; \frac{1}{Q(\hat{x})\, M} - 1, \tag{4.3}$$

where $\alpha = 1/M$. The tolerated bias is largest when $Q \approx P$, that is, when $M \to 1$, so VRS is most robust when the proposal is close to the target; conversely, if the proposal is far from the target, VRS can in principle underperform direct sampling.

With $q = 0.5$ fixed, we can compute the maximum tolerated bias c at each target p from Equation (4.3) and overlay it as a shaded $\pm c$ band on the direct sampling

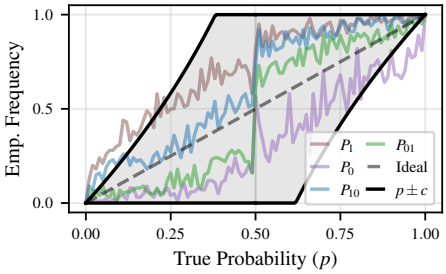


FIGURE 4.13: Direct-sampling calibration with shaded $\pm c$ boxes at each p, where c is the maximum tolerated bias under Corollary 1. Most observed frequencies fall inside the boxes. Figure copied from Xiao *et al.* [17].

calibration plot (Figure 4.13). Most of the empirically observed frequencies fall inside the band, meaning the inequality in Equation (4.3) is satisfied in practice. This supports the explanation that the rejection-sampling algorithm, not the VRS prompt, is what drives the improvement.

Discussion

VRS imports classical rejection sampling into the language interface, treating the LLM as a black-box sampler and keeping the recipe model agnostic. Across four diverse LLMs, open and proprietary, VRS substantially reduces calibration error. The theoretical analysis identifies the regime in which the improvement is provable rather than merely empirical. The algorithm, not the prompt, is key to its success. This is a useful corrective given how much recent work has invested in prompt engineering alone. The fact that language-only interventions such as chain-of-thought cannot close the gap implies *sampling is a fundamentally distinct ability from question answering*. Question answering tasks have per instance supervision, while i.i.d. sampling only has distribution level verifiability.

Faithful sampling also has practical stakes. Monte Carlo methods, agent-based simulations, and randomized decision making all rely on it. A user who sees an LLM accurately describe a distribution naturally trusts it to sample from that distribution; yet hidden bias can contaminate downstream workflows, skew survey simulators, or introduce unfairness in stochastic tie-breakers, raising safety, reliability, and fairness concerns across the stack. In the arc of the chapter, Section 4.1 proposed natural language as a computational medium, Section 4.2 built a language-based machine learning algorithm, and the present section builds a language-based sampler, demonstrating examples of adapting a classical computing task into natural language. More broadly, VRS points to a path for integrating principled randomness into LLM-based systems, with classical probabilistic tools verbalized and embedded into LLM workflows, delivering reliability without opaque prompt engineering.

5

DISCUSSION: TOWARD VERBALIZED COMPUTING

Large language models have made natural language a medium of computation. The medium is semantically unreliable, and the question is what principles can make computation in it reliable. This thesis has argued that the probabilistic toolkit developed for numerical computation is a natural starting point, and the five papers collected here take first steps in that direction. Chapter 2 laid out the technical background shared by the two paradigms. Chapter 3 developed the probabilistic toolkit in its home territory, first through an information-theoretic account of representation in hierarchical latent variable models, then through an empirical study of generalization in variational autoencoders. Chapter 4 carried representation, learning, and sampling across to natural language, first by drawing a structural parallel between language models and stored-program computers, then by parameterizing a learner entirely in text, and finally by adapting rejection sampling into a verbalized accept/reject procedure. In this closing chapter, we step back from the individual contributions, ask what organizing picture they suggest, and point to where the work goes next.

5.1 THE COMPLEXITY OF FORMALIZATION

The verbalized procedures of Chapter 4 operate at a small scale, on tasks that can reasonably be called toys. The simplicity is deliberate. These are starting points, chosen so that the mechanics of a verbalized procedure can be studied clearly before the method is scaled to harder problems. The question that follows is about the limits and the potential: when and why should verbalized computing matter on tasks that are not toys? The usual answers appeal to expressiveness, interpretability, or raw performance. None of these satisfies us. The answer we believe is truly fundamental comes from an overlooked but important perspective: the complexity of formalization.

Classical complexity theory measures the cost of solving a problem once it has been stated in a formal language. It does not measure the cost of getting the problem into that formal language in the first place. That step, translating a messy real-world question into variables, distributions, objectives, and constraints, is often the most expensive part in terms of human reasoning. We call this the cost of formalization, and argue that verbalized computing is interesting in large part because it does include the process of formalization into the computation, unlike classical computing.

A verbalized procedure takes as input a natural language description of the task, the data, and the acceptance criterion, and produces a decision or a sample without requiring the practitioner to write down a formal model. The verbalized

machine learning framework of Chapter 4 illustrates the shift at the level of model specification: the parameters and model family are natural language sentences rather than a weight vector or a fixed formula. Verbalized rejection sampling illustrates the shift at the level of algorithm specification: the accept/reject criterion is stated in words, and the sampler applies it without an explicit density. In both cases, the formalization step shrinks. In exchange, the guarantees weaken. The numerical counterparts come with convergence guarantees and theorems; the verbalized counterparts come with empirical performance on specific tasks and, at best, partial theoretical analysis of when the verbal procedure beats a baseline.

We propose the complexity of formalization as a lens for reasoning about when a verbalized approach is worth considering. The lens does not give a formal prescription; we do not yet know how to measure formalization cost in a way that is both rigorous and useful. It does suggest where to look. Problems for which the formalization step is expensive, such as open ended specification, problems with ill defined inputs, or problems whose specification changes during the task itself, are natural candidates for a verbalized procedure. Problems for which the formalization step is cheap, or has already been done once and can be reused, have less to gain. Whether this lens can be sharpened into a theory is the open question we leave for future work. The precedent is clear. Classical computation formalizes a program, and complexity theory measures the cost in machine steps. Deep learning approximates a function from data, and learning theory measures the cost in data. Verbalized computing specifies the task in natural language, and a theory of the complexity of formalization would play the same role for it, measuring the cost of getting a problem into a form the system can act on.

5.2 FUTURE DIRECTIONS

Verbalized deep learning

The verbalized machine learning framework of Chapter 4 parameterizes a model by a single natural language prompt and updates that prompt iteratively. In the language of classical machine learning, this is analogous to a single perceptron: one layer of computation, one parameter, one optimization loop. Whatever lessons one draws from the perceptron apply here as well. The expressivity of a single verbalized unit is limited, and the behaviours it cannot capture are numerous.

The obvious next step is scaling, a path the field has taken before. Processor performance improved by packing more transistors onto a die. The deep learning revolution came from stacking more neurons and more layers. If verbalized machine learning follows the same pattern, it will scale by connecting many reasoning units into a larger system. This is a scaling axis distinct from, and complementary to, improving the individual unit. A deep verbalized system of this kind would be built from small units, each with its own verbalized parameter, wired into an architecture that routes intermediate outputs between them. Training such a model means updating each unit's prompt in response to the behaviour of the whole. The

numerical analogue, back propagation through a multilayer network, has no direct verbalized counterpart, and finding one is among the most concrete open problems in verbalized machine learning. Whether the patterns that proved successful in deep learning, such as depth, residual connections, and attention, carry over to language space, or whether the right compositional primitives are different, is an empirical question that the present thesis does not settle.

The practical target of this direction is what is often called multi-agent systems. Many real world tasks, from operating a research group to running a company, are specified informally and are solved by organizing many smaller agents, each reasoning about a part of the problem. Current multi-agent systems rely on hand designed workflows that generalize poorly across tasks. A general theory of verbalized deep learning would replace those workflows with a principled way of composing units, trained rather than hand designed, and would make the number of agents a scaling dimension in its own right.

Verbalized probabilistic algorithms

Verbalized rejection sampling opens a broader program of uncertainty-aware reasoning in language. Classical probabilistic algorithms, from importance sampling and Markov chain Monte Carlo to Thompson sampling, are designed to produce stochastic outputs with specific guarantees: unbiased estimates, samples from a target distribution, or actions that accumulate near-optimal reward. Each has a natural verbalized counterpart. Whether these verbalized counterparts recover the same guarantees, and under what conditions, is open.

Reliability in language-based sampling has practical stakes beyond toy distributions. General purpose language agents that plan and act under uncertainty across many steps, such as OpenClaw [95], are being deployed for open ended tasks, and their decisions depend on how they sample from options, weigh alternatives, and update beliefs in response to evidence. A principled probabilistic layer for verbalized reasoning could give such systems the kind of calibration and provable sampling behaviour that numerical Monte Carlo gives numerical simulations.

Foundational questions

Two foundational questions sit above the contributions of this thesis.

The first concerns reliability. Shannon's channel coding theorem fixes the capacity of a noisy communication channel and tells us exactly how much information can be transmitted reliably across it. Von Neumann's threshold theorem fixes the error rate at which unreliable components can be combined into arbitrarily reliable circuits. Both are theorems about physical unreliability, noise in signals and failures in hardware. The noise that verbalized computing must handle is semantic, operating on meaning rather than on bits, and at present no analogous theory exists. Building one is the foundational reliability question of verbalized computing.

The second concerns expressive power. Every formal statement can be paraphrased in natural language, but some natural language statements resist clean formalization, such as underspecified goals, defeasible rules, and ambiguous predicates. Are there problems that can be stated and solved in natural language but not in any formal language? Are there problems that can be solved more efficiently in natural language than through a formalization step? In mathematical practice, an informal proof sketch can be convincing before a fully formal proof is written. Whether this is an illusion, a matter of definition, or a genuine computational property is open.

5.3 CONCLUDING REMARKS

This thesis is an early contribution to verbalized computing, a paradigm in which natural language serves as a medium of computation. The five papers develop the probabilistic toolkit in numerical machine learning, and show that representation, learning, and sampling admit verbalized counterparts in language. Several problems remain open, including a theory of formalization cost and an account of the limits of natural language as a computational medium. We hope the work presented here is a stepping stone toward a principled understanding of what verbalized computing can, and cannot, reliably do.

REFERENCES

1. Turing, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* **s2-42**, Article 230 (1937) (cit. on p. 1).
2. Turing, A. M. Computing Machinery and Intelligence. *Mind* **59**, 433 (1950) (cit. on pp. 1, 37).
3. Shannon, C. E. A Mathematical Theory of Communication. *The Bell System Technical Journal* **27**, 379 (1948) (cit. on p. 1).
4. Von Neumann, J. in *Automata Studies* 43 (1956) (cit. on p. 1).
5. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. *Language Models are Few-Shot Learners* in *Proceedings of Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H.) (2020) (cit. on pp. 2, 3, 18, 33, 34).
6. MacKay, D. J. *Information theory, inference and learning algorithms* (Cambridge university press, 2003) (cit. on pp. 2, 12, 16, 25).
7. Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association* (2017) (cit. on p. 2).
8. Kingma, D. P. & Welling, M. *Auto-Encoding Variational Bayes* in *International Conference on Learning Representations* (2014) (cit. on pp. 2, 13).
9. Neal, R. M. in *Handbook of Markov Chain Monte Carlo* 47 (2011) (cit. on p. 2).
10. Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A. & Murphy, K. *Fixing a broken ELBO* in *International Conference on Machine Learning* (2018), 159 (cit. on pp. 3, 14, 22, 24).
11. Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. *Understanding Deep Learning Requires Rethinking Generalization* in *International Conference on Learning Representations (ICLR)* (2017) (cit. on p. 3).
12. Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J. & Fedus, W. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research (TMLR)* **2022** (2022) (cit. on pp. 3, 19, 33, 34).

13. Xiao, T. Z. & Bamler, R. *Trading Information between Latents in Hierarchical Variational Autoencoders* in *International Conference on Learning Representations (ICLR)* (2023) (cit. on pp. 4, 6, 22–24, 26).

14. Xiao, T. Z., Zenn, J. & Bamler, R. *A Note on Generalization in Variational Autoencoders: How Effective Is Synthetic Data and Overparameterization?* in *Transactions on Machine Learning Research (TMLR)* (2024) (cit. on pp. 4, 6, 28–31).

15. Xiao, T. Z., Liu, W. & Bamler, R. Large Language Models Are Zero-Shot Problem Solvers—Just Like Modern Computers. *Harvard Data Science Review* (2025) (cit. on pp. 4, 6, 34–36).

16. Xiao, T. Z., Bamler, R., Schölkopf, B. & Liu, W. *Verbalized Machine Learning: Revisiting Machine Learning with Language Models* in *Transactions on Machine Learning Research (TMLR)* (2025) (cit. on pp. 5, 6, 38, 40–43).

17. Xiao, T. Z., Zenn, J., Liu, Z., Liu, W., Bamler, R. & Schölkopf, B. Flipping Against All Odds: Reducing LLM Coin Flip Bias via Verbalized Rejection Sampling. *arXiv preprint arXiv:2506.09998* (2025) (cit. on pp. 5, 6, 44–49).

18. Qiu, Z., Buchholz, S., Xiao, T. Z., Dax, M., Schölkopf, B. & Liu, W. *Reparameterized LLM Training via Orthogonal Equivalence Transformation* in *Advances in Neural Information Processing Systems (NeurIPS)* (2025) (cit. on p. 6).

19. Qiu, Z., Liu, W., Feng, H., Liu, Z., Xiao, T. Z., Collins, K. M., Tenenbaum, J. B., Weller, A., Black, M. J. & Schölkopf, B. *Can Large Language Models Understand Symbolic Graphics Programs?* in *International Conference on Learning Representations (ICLR)* (2025) (cit. on p. 6).

20. Liu, Z., Xiao, T. Z., Liu, W., Bengio, Y. & Zhang, D. *Fast Diversity-Preserving Reward Finetuning of Diffusion Models via Nabla-GFlowNets* in *International Conference on Learning Representations (ICLR)* (2025) (cit. on p. 6).

21. Ou, Z., Zhang, M., Zhang, A., Xiao, T. Z., Li, Y. & Barber, D. *Improving Probabilistic Diffusion Models With Optimal Covariance Matching* in *International Conference on Learning Representations (ICLR)* (2025) (cit. on p. 6).

22. Zhang, A., Xiao, T. Z., Liu, W., Bamler, R. & Wischik, D. *Your Finetuned Large Language Model is Already a Powerful Out-of-distribution Detector* in *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2025) (cit. on p. 7).

23. Yu, Z., Yuan, Y., Xiao, T. Z., Xia, F., Fu, J., Zhang, G., Lin, G. & Liu, W. *Generating Symbolic World Models via Test-time Scaling of Large Language Models* in *Transactions on Machine Learning Research (TMLR)* (2025) (cit. on p. 7).

24. Xiao, T. Z., Liu, W. & Bamler, R. A Compact Representation for Bayesian Neural Networks By Removing Permutation Symmetry. *NeurIPS 2023 Workshop on Unifying Representations in Neural Models* (2023) (cit. on p. 7).

25. Xiao, T. Z., Zenn, J. & Bamler, R. The SVHN Dataset Is Deceptive for Probabilistic Generative Models Due to a Distribution Mismatch. *NeurIPS 2023 Workshop on Distribution Shifts* (2023) (cit. on p. 7).

26. Qiu, Z., Liu, W., Xiao, T. Z., Liu, Z., Bhatt, U., Luo, Y., Weller, A. & Schölkopf, B. *Iterative Teaching by Data Hallucination* in *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2023) (cit. on p. 7).
27. Cover, T. M. & Thomas, J. A. *Elements of Information Theory* 2nd ed. (2006) (cit. on pp. 10, 11).
28. Kullback, S. & Leibler, R. A. On information and sufficiency. *The Annals of Mathematical Statistics* **22** (1951) (cit. on p. 11).
29. Rezende, D. J., Mohamed, S. & Wierstra, D. *Stochastic backpropagation and approximate inference in deep generative models* in *International Conference on Machine Learning* (2014) (cit. on p. 13).
30. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S. & Lerchner, A. *beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework* in *International Conference on Learning Representations* (2017) (cit. on p. 14).
31. Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K. & Winther, O. Ladder variational autoencoders. *Advances in Neural Information Processing Systems* **29** (2016) (cit. on pp. 14, 22, 23).
32. Maaløe, L., Fraccaro, M., Liévin, V. & Winther, O. Biva: A very deep hierarchy of latent variables for generative modeling. *Advances in Neural Information Processing Systems* **32** (2019) (cit. on pp. 14, 22).
33. Vahdat, A. & Kautz, J. NVAE: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems* **33**, 19667 (2020) (cit. on pp. 14, 22).
34. Child, R. *Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images* in *International Conference on Learning Representations* (2021) (cit. on pp. 14, 22).
35. Ho, J., Jain, A. & Abbeel, P. *Denoising diffusion probabilistic models* in *Advances in Neural Information Processing Systems* (2020) (cit. on pp. 15, 30).
36. Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S. & Poole, B. *Score-Based Generative Modeling through Stochastic Differential Equations* in *International Conference on Learning Representations* (2021) (cit. on pp. 15, 30).
37. Cremer, C., Li, X. & Duvenaud, D. *Inference suboptimality in variational autoencoders* in *International Conference on Machine Learning* (2018) (cit. on pp. 15, 28, 30).
38. Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B. & Sutskever, I. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment* **2021**, 124003 (2021) (cit. on pp. 15, 28, 31).
39. Belkin, M., Hsu, D., Ma, S. & Mandal, S. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* **116**, 15849 (2019) (cit. on pp. 15, 28, 31).

40. Robert, C. P. & Casella, G. *Monte Carlo Statistical Methods* 2nd ed. (2004) (cit. on p. 16).

41. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. *Attention is All you Need* in *Proceedings of Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, NIPS 2017* (eds Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N. & Garnett, R.) (2017), 5998 (cit. on pp. 17, 35).

42. Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. & Amodei, D. *Scaling laws for neural language models*. ArXiv (2020) (cit. on pp. 18, 34).

43. Fan, A., Lewis, M. & Dauphin, Y. *Hierarchical Neural Story Generation* in *Annual Meeting of the Association for Computational Linguistics (ACL)* (2018) (cit. on p. 18).

44. Holtzman, A., Buys, J., Du, L., Forbes, M. & Choi, Y. *The Curious Case of Neural Text Degeneration* in *International Conference on Learning Representations (ICLR)* (2020) (cit. on p. 18).

45. Xie, S. M., Raghunathan, A., Liang, P. & Ma, T. *An Explanation of In-context Learning as Implicit Bayesian Inference* in *International Conference on Learning Representations (ICLR)* (2022) (cit. on p. 18).

46. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., *et al. Chain-of-thought prompting elicits reasoning in large language models* in *NeurIPS* (2022) (cit. on pp. 19, 44).

47. Kojima, T., Gu, S. S., Reid, M., Matsuo, Y. & Iwasawa, Y. *Large Language Models are Zero-Shot Reasoners* in *Proceedings of Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022* (eds Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K. & Oh, A.) (2022) (cit. on pp. 19, 34, 36).

48. Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D. & Chen, X. *Large language models as optimizers* in *ICLR* (2024) (cit. on p. 19).

49. Gu, J., Pang, L., Shen, H. & Cheng, X. Do LLMs Play Dice? Exploring Probability Distribution Sampling in Large Language Models for Behavioral Simulation. *arXiv preprint arXiv:2404.09043* (2024) (cit. on pp. 19, 44).

50. Van Koevering, K. & Kleinberg, J. How Random is Random? Evaluating the Randomness and Humaness of LLMs' Coin Flips. *arXiv preprint arXiv:2406.00092* (2024) (cit. on pp. 20, 44).

51. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S. & Lerchner, A. *beta-vae: Learning basic visual concepts with a constrained variational framework* in *International Conference on Learning Representations* (2017) (cit. on p. 22).

52. Tishby, N. & Zaslavsky, N. *Deep learning and the information bottleneck principle* in *2015 ieee information theory workshop (itw)* (2015), 1 (cit. on p. 22).

53. Alemi, A. A., Fischer, I., Dillon, J. V. & Murphy, K. *Deep Variational Information Bottleneck* in *International Conference on Learning Representations* (2016) (cit. on p. 22).

54. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B. & Ng, A. Y. *Reading Digits in Natural Images with Unsupervised Feature Learning* in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2011) (cit. on p. 24).

55. Krizhevsky, A., Hinton, G., *et al.* Learning multiple layers of features from tiny images (2009) (cit. on pp. 24, 30).

56. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* (1998) (cit. on p. 24).

57. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. & Chen, X. Improved techniques for training gans. *Advances in Neural Information Processing Systems* **29** (2016) (cit. on p. 24).

58. Meyen, S. *Relation between classification accuracy and mutual information in equally weighted classification tasks* MA thesis (Universität Hamburg, 2016) (cit. on p. 25).

59. Shu, R., Bui, H. H., Zhao, S., Kochenderfer, M. J. & Ermon, S. *Amortized inference regularization* in *Advances in Neural Information Processing Systems* (2018) (cit. on pp. 28, 30).

60. Kuzina, A., Welling, M. & Tomczak, J. *Alleviating adversarial attacks on variational autoencoders with mcmc* in *Advances in Neural Information Processing Systems* (2022) (cit. on p. 29).

61. Wang, Z., Simoncelli, E. P. & Bovik, A. C. *Multiscale structural similarity for image quality assessment* in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003* (2003) (cit. on p. 29).

62. Alemohammad, S., Casco-Rodriguez, J., Luzi, L., Humayun, A. I., Babaei, H., LeJeune, D., Siahkoohi, A. & Baraniuk, R. *Self-Consuming Generative Models Go MAD* in *International Conference on Learning Representations* (2024) (cit. on pp. 30, 32).

63. Shumailov, I., Shumaylov, Z., Zhao, Y., Gal, Y., Papernot, N. & Anderson, R. The Curse of Recursion: Training on Generated Data Makes Models Forget. *arXiv preprint arxiv:2305.17493* (2023) (cit. on pp. 30, 32).

64. Karras, T., Aittala, M., Aila, T. & Laine, S. *Elucidating the Design Space of Diffusion-Based Generative Models* in *Advances in Neural Information Processing Systems* (2022) (cit. on p. 30).

65. Wu, Y., Burda, Y., Salakhutdinov, R. & Grosse, R. *On the Quantitative Analysis of Decoder-Based Generative Models* in *International Conference on Learning Representations* (2017) (cit. on p. 30).

66. Curth, A., Jeffares, A. & van der Schaar, M. A u-turn on double descent: Rethinking parameter counting in statistical learning. *arXiv preprint arXiv:2310.18988* (2023) (cit. on p. 31).

67. Wittgenstein, L. *Tractatus Logico-Philosophicus* trans. by Ogden, C. K. (Routledge & Kegan Paul, London, 1922) (cit. on p. 33).

68. Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M., Ali, M., Yang, Y. & Zhou, Y. *Deep learning scaling is predictable, empirically*. ArXiv (2017) (cit. on p. 34).

69. Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., El Showk, S., Fort, S., Hatfield-Dodds, Z., Henighan, T., Johnston, S., Jones, A., Joseph, N., Kernian, J., Kravec, S., Mann, B., Nanda, N., Ndousse, K., Olsson, C., Amodei, D., Brown, T., Kaplan, J., McCandlish, S., Olah, C., Amodei, D. & Clark, J. *Predictability and surprise in large generative models* in *Proceedings of 2022 ACM Conference on Fairness, Accountability, and Transparency* (2022), 1747 (cit. on p. 34).

70. Schulman, J., Zoph, B., Kim, C., Hilton, J., Menick, J., Weng, J., Uribe, J. F. C., Fedus, L., Metz, L., Pokorny, M., Lopes, R. G., Zhao, S., Vijayvergiya, A., Sigler, E., Perelman, A., Voss, C., Heaton, M., Parish, J., Cummings, D., Nayak, R., Balcom, V., Schnurr, D., Kaftan, T., Hallacy, C., Turley, N., Deutsch, N., Goel, V., Ward, J., Konstantinidis, A., Zaremba, W., Ouyang, L., Bogdonoff, L., Gross, J., Medina, D., Yoo, S., Lee, T., Lowe, R., Mossing, D., Huizinga, J., Jiang, R., Wainwright, C., Almeida, D., Lin, S., Zhang, M., Xiao, K., Slama, K., Bills, S., Gray, A., Leike, J., Pachocki, J., Tillet, P., Jain, S., Brockman, G. & Ryder, N. ChatGPT: Optimizing Language Models for Dialogue. *OpenAI* (2022, November 30) (cit. on p. 34).

71. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog* (2019) (cit. on pp. 34, 35).

72. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S. & Fiedel, N. *PaLM: Scaling language modeling with pathways*. ArXiv (2022) (cit. on p. 34).

73. Anderson, P. W. More Is Different. *Science* **177**, 393 (1972) (cit. on p. 34).

74. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* **79**, 2554 (1982) (cit. on p. 34).

75. Weik, M. H. The ENIAC story. *Ordnance* **45**, 571 (1961) (cit. on p. 35).

76. Von Neumann, J. *First draft of a report on the EDVAC*. University of Pennsylvania (1945) (cit. on p. 35).

77. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E. & Dean, J. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer* in *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017* (OpenReview.net, 2017) (cit. on p. 35).

78. Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L. & Uszkoreit, J. *One model to learn them all*. ArXiv (2017) (cit. on p. 35).

79. McCann, B., Keskar, N. S., Xiong, C. & Socher, R. *The natural language decathlon: Multitask learning as question answering*. ArXiv (2018) (cit. on p. 35).

80. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J. & Lowe, R. *Training language models to follow instructions with human feedback* in *Proceedings of Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022* (eds Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K. & Oh, A.) (2022) (cit. on p. 36).

81. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H. & Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys* **55**, 1 (2023) (cit. on p. 36).

82. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., *et al.* Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023) (cit. on p. 39).

83. Yang, J., Shi, R., Wei, D., Liu, Z., Zhao, L., Ke, B., Pfister, H. & Ni, B. MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Scientific Data* (2023) (cit. on p. 40).

84. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., *et al.* Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023) (cit. on pp. 40, 44).

85. Pryzant, R., Iter, D., Li, J., Lee, Y. T., Zhu, C. & Zeng, M. *Automatic Prompt Optimization with" Gradient Descent" and Beam Search* in *EMNLP* (2023) (cit. on p. 42).

86. Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C. & Zou, J. TextGrad: Automatic "Differentiation" via Text. *arXiv preprint arXiv:2406.07496* (2024) (cit. on p. 42).

87. Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H. & Ba, J. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022) (cit. on p. 42).

88. Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., *et al.* The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024) (cit. on p. 44).

89. Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., *et al.* Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024) (cit. on p. 44).

90. Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., *et al.* Qwen2. 5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024) (cit. on p. 44).

91. Gupta, R., Corona, R., Ge, J., Wang, E., Klein, D., Darrell, T. & Chan, D. M. Enough Coin Flips Can Make LLMs Act Bayesian. *arXiv preprint arXiv:2503.04722* (2025) (cit. on p. 44).

92. Meister, N., Guestrin, C. & Hashimoto, T. Benchmarking Distributional Alignment of Large Language Models. *arXiv preprint arXiv:2411.05403* (2024) (cit. on p. 44).

93. Hopkins, A. K. & Renda, A. *Can llms generate random numbers? evaluating llm sampling in controlled domains* in *Sampling and Optimization in Discrete Space (SODS) ICML 2023 Workshop* (2023) (cit. on p. 44).

94. Bar-Hillel, M., Peer, E. & Acquisti, A. "Heads or tails?"—A reachability bias in binary choice. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **40**, 1656 (2014) (cit. on p. 45).

95. Steinberger, P. *OpenClaw* `https://github.com/openclaw/openclaw`. Accessed 2026-04-24. 2026 (cit. on p. 53).

APPENDIX

A PUBLICATIONS

This thesis includes the following articles which have been published in peer-reviewed journals or conferences:

1. T. Z. Xiao & R. Bamler, Trading Information between Latents in Hierarchical Variational Autoencoders. *International Conference on Learning Representations (ICLR)* (2023).

2. T. Z. Xiao*, J. Zenn* & R. Bamler, A Note on Generalization in Variational Autoencoders: How Effective Is Synthetic Data and Overparameterization? *Transactions on Machine Learning Research (TMLR)* (2024).

3. T. Z. Xiao, W. Liu & R. Bamler, Large Language Models Are Zero-Shot Problem Solvers — Just Like Modern Computers. *Harvard Data Science Review* (2025).

4. T. Z. Xiao, R. Bamler, B. Schölkopf & W. Liu, Verbalized Machine Learning: Revisiting Machine Learning with Language Models. *Transactions on Machine Learning Research (TMLR)* (2025).

In addition, it includes the following pre-print article:

5. T. Z. Xiao, J. Zenn, Z. Liu, W. Liu, R. Bamler & B. Schölkopf, Flipping Against All Odds: Reducing LLM Coin Flip Bias via Verbalized Rejection Sampling. *arXiv preprint arXiv:2506.09998* (2025).

* indicates shared first authorship.

Below, we provide copies of these publications and the pre-print article.

Published as a conference paper at ICLR 2023

TRADING INFORMATION BETWEEN LATENTS IN HIERARCHICAL VARIATIONAL AUTOENCODERS

Tim Z. Xiao
University of Tübingen & IMPRS-IS
zhenzhong.xiao@uni-tuebingen.de

Robert Bamler
University of Tübingen
robert.bamler@uni-tuebingen.de

ABSTRACT

Variational Autoencoders (VAEs) were originally motivated (Kingma & Welling, 2014) as probabilistic generative models in which one performs approximate Bayesian inference. The proposal of β-VAEs (Higgins et al., 2017) breaks this interpretation and generalizes VAEs to application domains beyond generative modeling (e.g., representation learning, clustering, or lossy data compression) by introducing an objective function that allows practitioners to trade off between the information content ("bit rate") of the latent representation and the distortion of reconstructed data (Alemi et al., 2018). In this paper, we reconsider this rate/distortion trade-off in the context of hierarchical VAEs, i.e., VAEs with more than one layer of latent variables. We identify a general class of inference models for which one can split the rate into contributions from each layer, which can then be tuned independently. We derive theoretical bounds on the performance of downstream tasks as functions of the individual layers' rates and verify our theoretical findings in large-scale experiments. Our results provide guidance for practitioners on which region in rate-space to target for a given application.

1 INTRODUCTION

Variational autoencoders (VAEs) (Kingma & Welling, 2014; Rezende et al., 2014) are a class of deep generative models that are used, e.g., for density modeling (Takahashi et al., 2018), clustering (Jiang et al., 2017), nonlinear dimensionality reduction of scientific measurements (Laloy et al., 2017), data compression (Ballé et al., 2017), anomaly detection (Xu et al., 2018), and image generation (Razavi et al., 2019). VAEs (more precisely, β-VAEs (Higgins et al., 2017)) span such a diverse set of application domains in part because they can be tuned to a specific task without changing the network architecture, in a way that is well understood from information theory (Alemi et al., 2018).

The original proposal of VAEs (Kingma & Welling, 2014) motivates them from the perspective of generative probabilistic modeling and approximate Bayesian inference. However, the generalization to β-VAEs breaks this interpretation as they are no longer trained by maximizing a lower bound on the marginal data likelihood. These models are better described as neural networks that are trained to learn the identity function, i.e., to make their output resemble the input as closely as possible. This task is made nontrivial by introducing a so-called (variational) information bottleneck (Alemi et al., 2017; Tishby & Zaslavsky, 2015) at one or more layers, which restricts the information content that passes through these layers. The network activations at the information bottleneck are called latent representations (or simply "latents"), and they split the network into an encoder part (from input to latents) and a decoder part (from latents to output). This separation of the model into an encoder and a decoder allows us to categorize the wide variety of applications of VAEs into three domains:

1. **data reconstruction tasks**, i.e., applications that involve *both the encoder and the decoder*; these include various nonlinear inter- and extrapolations (e.g., image upscaling, denoising, or inpainting), and VAE-based methods for lossy data compression;
2. **representation learning tasks**, i.e., applications that involve *only the encoder*; they serve a downstream task that operates on the (typically lower dimensional) latent representation, e.g., classification, regression, visualization, clustering, or anomaly detection; and
3. **generative modeling tasks**, i.e., applications that involve *only the decoder* are less common but include generating new samples that resemble training data.

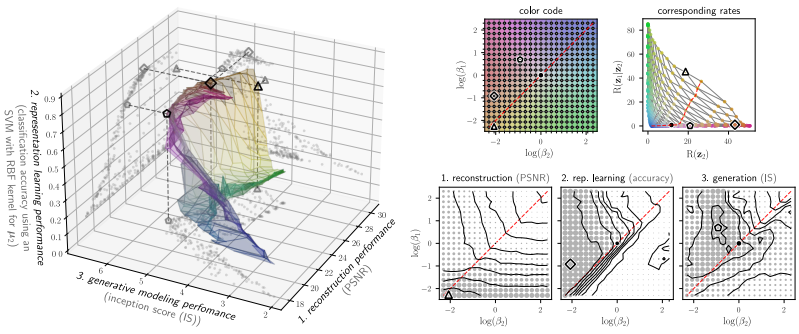


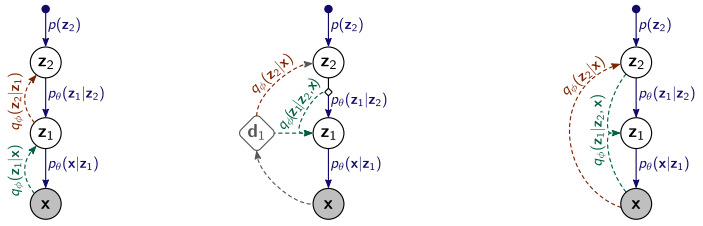
Figure 1: Left: trade-off between performance in the three applications domains of VAEs, using GHVAE trained on the SVHN data set (details: Section 5); higher is better for all three metrics; gray dots on walls show 2d-projections. Right: color code, corresponding layer-wise rates (Eq. 7), and individual performance landscapes (size of dots $\propto$ performance). The hyperparameters β_2 and β_1 allow us to tune the HVAE for best data reconstruction ($\triangle$), best representation learning ($\diamond$), or best generative modeling ($\pentagon$). Note that performance landscapes differ strongly across the three applications, and neither a standard VAE ($\beta_2 = \beta_1 = 1$; marked "$\bullet$" in right panels) nor a conventional β-VAE ($\beta_2 = \beta_1$; dashed red lines) result in optimal models for any of the three applications.

The information bottleneck incentivizes the VAE to encode information into the latents efficiently by removing any redundancies from the input. How agressively this is done can be controlled by tuning the strength β of the information bottleneck (Alemi et al., 2018). Unfortunately, information theory distinguishes relevant from redundant information only in a quantitative way that is agnostic to the qualitative features that each piece of information represents about some data point. In practice, many VAE-architectures (Deng et al., 2017; Yingzhen & Mandt, 2018; Ballé et al., 2018) try to separate qualitatively different features into different parts of the latent representation by making the model architecture reflect some prior assumptions about the semantic structure of the data. This allows downstream applications from the three domains discussed above to more precisely target specific qualitative aspects of the data by using or manipulating only the corresponding part of the latent representation. However, in this approach, the degree of detail to which each qualitative aspect is encoded in the latents can be controlled at most indirectly by tuning network layer sizes.

In this paper, we argue both theoretically and empirically that the three different application domains of VAEs identified above require different trade-offs in the amount of information that is encoded in each part of the latent representation. We propose a method to independently control the information content (or "rate") of each layer of latent representations, generalizing the rate/distortion theory of β-VAEs (Alemi et al., 2018) for VAEs with more than one layer of latents ("hierarchical VAEs" or HVAEs for short). We identify the most general model architecture that is compatible with our proposal and analyze how both theoretical performance bounds and empirically measured performances in each of the above three application domains depend on how rate is distributed across layers.

Our approach is summarized in Figure 1. The 3d-plot shows empirically measured performance metrics (discussed in detail in Section 5.2) for the three application domains identified above. Each point on the colored surface corresponds to different layer-wise rates in an HVAE with two layers of latents. Crucially, the rates that lead to optimal performance are different for each of the three application domains (see markers $\triangle$, $\pentagon$, and $\diamond$ in Figure 1), and none of these three optimal models coincide with a conventional β-VAE (dashed red lines in right panels). Thus, being able to control each layer's individual rate allows practitioners to train VAEs that target a specific application.

The paper is structured as follows. Section 2 summarizes related work. Section 3 introduces the proposed information-trading method. We then analyze how controlling individual layers' rates can be used to tune HVAEs for specific tasks, i.e., how performance in each of the three application domains identified above depends on the allocation of rates across layers. This analysis is done theoretically in Section 4 and empirically in Section 5. Section 6 provides concluding remarks.



(a) bottom-up (b) implicit top-down (e.g., LVAE) (c) generalized (explicit) top-down

Figure 2: Inference (dashed arrows) and generative (solid arrows) models for hierarchical VAEs (HVAEs) with two layers of latent variables. White/gray circles denote latent/observed random variables, respectively; the diamond $\mathbf{d}_1$ in (b) is the result of a deterministic transformation of $\mathbf{x}$.

2 Related Work

We group related work into work on model architectures for hierarchical VAEs, and on β-VAEs.

Model Design for Hierarchical VAEs. The original VAE design (Kingma & Welling, 2014; Rezende et al., 2014) has a single layer of latent variables, but recent works (Vahdat & Kautz, 2020; Child, 2021), found that increasing the number of stochastic layers in hierarchical VAEs (HVAEs) improves performance. HVAEs have various designs for their inference models. Sønderby et al. (2016) introduced Ladder VAE (LVAE) with a top-down inference path rather than the naive bottom-up inference (see Section 3), whereas the Bidirectional-Inference VAE (BIVA) (Maaløe et al., 2019) uses a combination of top-down and bottom-up. Our proposed framework applies to a large class of inference models (see Section 3) that includes the popular LVAE (Sønderby et al., 2016).

β-VAEs And Their Information-Theoretical Interpretations. Higgins et al. (2017) introduced an extra hyperparameter β in the objective of VAEs that tunes the strength of the information bottleneck, and they observed that large β leads to a disentangled latent representation. An information-theoretical interpretation of β-VAEs was provided in (Alemi et al., 2018) by applying the concept of a (variational) bottleneck (Tishby & Zaslavsky, 2015; Alemi et al., 2017) to autoencoders. Due to this information-theoretical interetation, β-VAEs are popular models for data compression (Ballé et al., 2017; Minnen et al., 2018; Yang et al., 2020), where tuning β allows trading off between the bit rate of compressed data and data distortion. In the present work, we generalize β-VAEs when applied to HVAEs, and we introduce a framework for tuning the rate of each latent layer individually.

3 A Hierarchical Information Trading Framework

We propose a refinement of the rate/distortion theory of β-VAEs (Alemi et al., 2018) that admits controlling individual layers' rates in VAEs with more than one layers of latents (hierarchical VAEs).

3.1 Conventional β-VAE With Hierarchical Latent Representations

We consider a hierarchical VAE (HVAE) for data $\boldsymbol{x}$ with L layers of latent representations $\{\boldsymbol{z}_\ell\}_{\ell=1}^L$. Figure 2, discussed further in Section 3.2 below, illustrates various model architectures for the example of $L = 2$. Solid arrows depict the generative model $p_\theta(\{\boldsymbol{z}_\ell\}, \boldsymbol{x})$, where θ are model parameters (neural network weights). We assume that the implementation factorizes $p_\theta(\{\boldsymbol{z}_\ell\}, \boldsymbol{x})$ as follows,

$$p_\theta(\{\boldsymbol{z}_\ell\}, \boldsymbol{x}) = p_\theta(\boldsymbol{z}_L)\, p_\theta(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L)\, p_\theta(\boldsymbol{z}_{L-2}|\boldsymbol{z}_{L-1}, \boldsymbol{z}_L) \cdots p_\theta(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2})\, p_\theta(\boldsymbol{x}|\boldsymbol{z}_{\geq 1}) \qquad (1)$$

where the notation $\boldsymbol{z}_{\geq n}$ for any n is short for the collection of latents $\{\boldsymbol{z}_\ell\}_{\ell=n}^L$ (thus, $\boldsymbol{z}_{\geq 1}$ and $\{\boldsymbol{z}_\ell\}$ are synonymous), and the numbering of latents from L down to 1 follows the common convention in the literature (Sønderby et al., 2016; Gulrajani et al., 2017; Child, 2021). The loss function of a

normal β-VAE (Higgins et al., 2017) with this generic architecture would be

$$\mathcal{L}_\beta(\theta,\phi) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{X}_{\text{train}}}\big[\underbrace{\mathbb{E}_{q_\phi(\{\boldsymbol{z}_\ell\}|\boldsymbol{x})}\big[-\log p_\theta(\boldsymbol{x}|\{\boldsymbol{z}_\ell\})\big]}_{=\text{“distortion” } D} + \beta\underbrace{D_{\mathrm{KL}}\big[q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x})\,\big\|\,p_\theta(\{\boldsymbol{z}_\ell\})\big]}_{=\text{“rate” } R}\big]. \tag{2}$$

Here, $q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x})$ is the inference (or “encoder”) model with parameteres ϕ, $\mathbb{X}_{\text{train}}$ is the training set, $D_{\mathrm{KL}}[\,\cdot\,\|\,\cdot\,]$ denotes Kullback-Leibler divergence, and the Lagrange parameter $\beta > 0$ trades off between a (total) rate R and a distortion D (Alemi et al., 2018). Setting $\beta = 1$ turns Eq. 2 into the negative ELBO objective of a regular VAE (Kingma & Welling, 2014). The rate R obtains its name as it measures the (total) information content that q_ϕ encodes into the latent representations $\{\boldsymbol{z}_\ell\}$, which would manifest itself in the expected bit rate when one optimally encodes a random draw $\{\boldsymbol{z}_\ell\} \sim q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x})$ using $p_\theta(\{\boldsymbol{z}_\ell\})$ as an entropy model (Agustsson & Theis, 2020; Bennett et al., 2002). An important observation pointed out in (Alemi et al., 2017) is that, regardless how rate R is traded off against distortion D by tuning β, their sum $R + D$ is—in expectation under any data distribution $p_{\text{data}}(\boldsymbol{x})$—always lower bounded by the entropy $H[p_{\text{data}}(\boldsymbol{x})] := \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[-\log p_{\text{data}}(\boldsymbol{x})]$,

$$\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[R + D] \geq H[p_{\text{data}}(\boldsymbol{x})] \qquad \forall\, p_{\text{data}}. \tag{3}$$

Limitations. The rate R in Eq. 2 is a property of the *collection* $\{\boldsymbol{z}_\ell\}$ of all latents, which can limit its interpretability for some inference models. For example, the common convention of enumerating layers $\boldsymbol{z}_\ell$ from $\ell = L$ down to 1 in Eq. 1 is reminiscent of a naive architecture for the inference model that factorizes in reverse order compared to Eq. 1 (“bottom up”, see dashed arrows in Figure 2(a)), i.e., $q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x}) = q_\phi(\boldsymbol{z}_1|\boldsymbol{x})\,q_\phi(\boldsymbol{z}_2|\boldsymbol{z}_1)\cdots q_\phi(\boldsymbol{z}_L|\boldsymbol{z}_{L-1})$. Using a HVAE with such a “bottom-up” inference model to reconstruct some given data point $\boldsymbol{x}$ would map $\boldsymbol{x}$ to $\boldsymbol{z}_1$ using $q_\phi(\boldsymbol{z}_1|\boldsymbol{x})$ and then map $\boldsymbol{z}_1$ back to the data space using $p_\theta(\boldsymbol{x}|\boldsymbol{z}_1)$, thus ignoring all latents $\boldsymbol{z}_\ell$ with $\ell > 1$. Yet, the rate term in Eq. 2 still depends on all latents, including the ones not needed to reconstruct any data (practical VAE-based compression methods using bits-back coding (Frey & Hinton, 1997) would, however, indeed use $\boldsymbol{z}_\ell$ with $\ell > 1$ as auxiliary variables for computational efficiency).

3.2 Trading Information Between Latents

Many HVAEs used in the literature allow us to resolve the limitations identified in Section 3.1. For example, the popular LVAE architecture (Sønderby et al., 2016), (Figure 2(b)), uses an inference model (dashed arrows) that traverses the latents $\{\boldsymbol{z}_\ell\}$ in the same order as the generative model (solid arrows). We consider the following generalization of this architecture (see Figure 2(c)),

$$q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x}) = q_\phi(\boldsymbol{z}_L|\boldsymbol{x})\,q_\phi(\boldsymbol{z}_{L-1}\,|\,\boldsymbol{z}_L,\boldsymbol{x})\,q_\phi(\boldsymbol{z}_{L-2}\,|\,\boldsymbol{z}_{L-1},\boldsymbol{z}_L,\boldsymbol{x})\;\cdots\;q_\phi(\boldsymbol{z}_1\,|\,\boldsymbol{z}_{\geq 2},\boldsymbol{x}). \tag{4}$$

Formally, Eq. 4 is just the product rule of probability theory and therefore holds for arbitrary inference models $q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x})$. More practically, however, we make the assumption that the actual implementation of $q_\phi(\{\boldsymbol{z}_\ell\}\,|\,\boldsymbol{x})$ follows the structure in Eq. 4. This means that, using the trained model, the most efficient way to map a given data point $\boldsymbol{x}$ to its reconstruction $\hat{\boldsymbol{x}}$ now involves *all* latents $\boldsymbol{z}_\ell$ (either drawing a sample or taking the mode at each step):

$$\boldsymbol{x} \xrightarrow{q_\phi(\boldsymbol{z}_L|\boldsymbol{x})} \boldsymbol{z}_L \xrightarrow{q_\phi(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L,\boldsymbol{x})} \boldsymbol{z}_{L-1} \longrightarrow \cdots \longrightarrow \boldsymbol{z}_2 \xrightarrow{q_\phi(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2},\boldsymbol{x})} \boldsymbol{z}_1 \xrightarrow{p_\theta(\boldsymbol{x}|\{\boldsymbol{z}_\ell\})} \hat{\boldsymbol{x}}. \tag{5}$$

Layer-wise Rates. We can interpret Eq. 5 in that it first maps $\boldsymbol{x}$ to a “crude” representation $\boldsymbol{z}_L$, which gets iteratively refined to $\boldsymbol{z}_1$, and finally to a reconstruction $\hat{\boldsymbol{x}}$. Note that each factor $q_\phi(\boldsymbol{z}_\ell\,|\,\boldsymbol{z}_{\geq\ell+1},\boldsymbol{x})$ of the inference model in Eq. 4 is conditioned not only on the previous layers $\boldsymbol{z}_{\geq\ell+1}$ but also on the original data $\boldsymbol{x}$. This allows the inference model to target each refinement step in Eq. 5 such that the reconstruction $\hat{\boldsymbol{x}}$ becomes close to $\boldsymbol{x}$. More formally, we chose the inference architecture in Eq. 4 such that it factorizes over $\{\boldsymbol{z}_\ell\}$ in the same order as the generative model (Eq. 1). This allows us to split the total rate R into a sum of layer-wise rates as follows,

$$\begin{aligned} R &= \mathbb{E}_{q_\phi(\{\boldsymbol{z}_\ell\}|\boldsymbol{x})}\left[\log\frac{q_\phi(\boldsymbol{z}_L|\boldsymbol{x})}{p_\theta(\boldsymbol{z}_L)} + \log\frac{q_\phi(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L,\boldsymbol{x})}{p_\theta(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L)} + \ldots + \log\frac{q_\phi(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2},\boldsymbol{x})}{p_\theta(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2})}\right] \\ &= R(\boldsymbol{z}_L) + R(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L) + R(\boldsymbol{z}_{L-2}\,|\,\boldsymbol{z}_{L-1},\boldsymbol{z}_L) + \;\ldots\; + R(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2}). \end{aligned} \tag{6}$$

Here,

$$\begin{aligned} R(\boldsymbol{z}_L) &= D_{\mathrm{KL}}\big[q_\phi(\boldsymbol{z}_L|\boldsymbol{x})\,\big\|\,p_\theta(\boldsymbol{z}_L)\big] \qquad \text{and} \\ R(\boldsymbol{z}_\ell|\boldsymbol{z}_{\geq\ell+1}) &= \mathbb{E}_{q(\boldsymbol{z}_{\geq\ell+1}|\boldsymbol{x})}\big[D_{\mathrm{KL}}\big[q_\phi(\boldsymbol{z}_\ell\,|\,\boldsymbol{z}_{\geq\ell+1},\boldsymbol{x})\,\big\|\,p_\theta(\boldsymbol{z}_\ell\,|\,\boldsymbol{z}_{\geq\ell+1})\big]\big] \end{aligned} \tag{7}$$

quantify the information content of the highest-order latent representation $\boldsymbol{z}_L$ and the (expected) *increase* in information content in each refinement step $\boldsymbol{z}_{\ell+1} \to \boldsymbol{z}_\ell$ in Eq. 5, respectively.

Controlling Each Layer's Rate. Using Eqs. 6-7, we generalize the rate/distortion trade-off from Section 3.1 by introducing L individual Lagrange multipliers $\beta_L, \beta_{L-1}, \ldots, \beta_1$, collectively denoted as boldface $\boldsymbol{\beta}$. This leads to a new loss function that generalizes Eq. 2 as follows,

$$\mathcal{L}_{\boldsymbol{\beta}}(\theta, \phi) = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{X}_{\text{train}}}\big[D + \beta_L R(\boldsymbol{z}_L) + \beta_{L-1} R(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L) + \ldots + \beta_1 R(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2})\big]. \tag{8}$$

Setting all βs to the same value recovers the conventional β-VAE (Eq. 2), which trades off distortion against *total* information content in $\{\boldsymbol{z}_\ell\}$. Tuning each β-hyperparameter individually allows trading off information content across latents. (In a very deep HVAE (i.e., large L) it may be more practical to group layers into only few bins and to use the same β-value for all layers within a bin.) We analyze how to tune βs for various applications theoretically in Section 4 and empirically in Section 5.

4 Information-Theoretical Performance Bounds for HVAEs

In this section, we analyze theoretically how various performance metrics for HVAEs are restricted by the individual layers' rates $R(\boldsymbol{z}_L)$ and $R(\boldsymbol{z}_\ell|\boldsymbol{z}_{\geq \ell+1})$ identified in Eq. 7 for a HVAE with "top-down" inference model. Our analysis motivates the use of the information-trading loss function in Eq. 8 for training HVAEs, following the argument from the introduction that VAEs are commonly used for a vast variety of tasks. As we show, different tasks require different trade-offs that can be targeted by tuning the Lagrange multipliers $\boldsymbol{\beta}$ in Eq. 8. We group tasks into the application domains of (i) data reconstruction and manipulation, (ii) representation learning, and (iii) data generation.

Data Reconstruction and Manipulation. The most obvious class of application domains of VAEs includes tasks that combine encoder and decoder to map some data point $\boldsymbol{x}$ to representations $\{\boldsymbol{z}_\ell\}$ and then back to the data space. The simplest performance metric for such data reconstruction tasks is the expected distortion $E_{p_{\text{data}}(\boldsymbol{x})}[D]$, which we can bound by combining Eq. 3 with Eqs. 6-7,

$$\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[D] \geq H[p_{\text{data}}(\boldsymbol{x})] - \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\big[R(\boldsymbol{z}_L) + R(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L) + \cdots + R(\boldsymbol{z}_1|\boldsymbol{z}_{\geq 2})\big]. \tag{9}$$

Eq. 9 would suggest that higher rates (i.e., lower β's) are always better for data reconstruction tasks. However, in many practical tasks (e.g., image upscaling, denoising, or inpainting) the goal is not solely to reconstruct the original data but also to manipulate the latent representations $\{\boldsymbol{z}_\ell\}$ in a meaningful way. Here, lower rates can lead to more semantically meaningful representation spaces (see, e.g., Section 5.6 below). Controlling how rate is distributed across layers via Eq. 8 may allow practitioners to have a semantically meaningful high-level representation $\boldsymbol{z}_L$ with low rate $R(\boldsymbol{z}_L)$ while still retaining a high *total* rate R, thus allowing for low distortion D without violating Eq. 9.

Representation Learning. In many practical applications, VAEs are used as nonlinear dimensionality reduction methods to prepare some complicated high-dimensional data $\boldsymbol{x}$ for downstream tasks such as classification, regression, visualization, clustering, or anomaly detection. We consider a classifier $p_{\text{cls.}}(y|\boldsymbol{z}_\ell)$ operating on the latents $\boldsymbol{z}_\ell$ at some level ℓ. We assume that the (unknown) true data generative process $p_{\text{data}}(y, \boldsymbol{x}) = p_{\text{data}}(y)\, p_{\text{data}}(\boldsymbol{x}|y)$ generates data $\boldsymbol{x}$ conditioned on some true label y, thus defining a Markov chain $y \xrightarrow{p_{\text{data}}} \boldsymbol{x} \xrightarrow{q_\phi} \boldsymbol{z}_\ell \xrightarrow{p_{\text{cls.}}} \hat{y}$ where $\hat{y} := \arg\max_y p_{\text{cls.}}(y|\boldsymbol{z}_\ell)$. Classification accuracy is bounded (Meyen, 2016) by a function of the mutual information $I_q(y; \boldsymbol{z}_\ell)$,

$$\begin{aligned}
I_q(y; \boldsymbol{z}_\ell) \leq I_q(\boldsymbol{x}; \boldsymbol{z}_\ell) &\equiv \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\left[\mathbb{E}_{q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})}\left[\log \frac{q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})}{q_\phi(\boldsymbol{z}_\ell)}\right]\right] \\
&= \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\left[\mathbb{E}_{q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})}\left[\log \frac{q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})}{p_\theta(\boldsymbol{z}_\ell)}\right]\right] - D_{\text{KL}}\big[q_\phi(\boldsymbol{z}_\ell) \,\big\|\, p_\theta(\boldsymbol{z}_\ell)\big] \\
&\leq \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\Bigg[\mathbb{E}_{q_\phi(\boldsymbol{z}_{\geq \ell}|\boldsymbol{x})}\left[\log \frac{q_\phi(\boldsymbol{z}_{\geq \ell}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}_{\geq \ell})}\right] \\
&\qquad - \mathbb{E}_{q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})}\Big[D_{\text{KL}}\big[q_\phi(\boldsymbol{z}_{\geq \ell+1} \,|\, \boldsymbol{x}, \boldsymbol{z}_\ell) \,\big\|\, p_\theta(\boldsymbol{z}_{\geq \ell+1}|\boldsymbol{z}_\ell)\big]\Big]\Bigg] \\
&\leq \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}\big[\underbrace{R(\boldsymbol{z}_L) + R(\boldsymbol{z}_{L-1}|\boldsymbol{z}_L) + \ldots + R(\boldsymbol{z}_\ell \,|\, \boldsymbol{z}_{\geq \ell+1})}_{=:R(\boldsymbol{z}_{\geq \ell})\ (\leq R)}\big].
\end{aligned} \tag{10}$$

Here, $q_\phi(\boldsymbol{z}_\ell) := \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})]$ and we identify $R(\boldsymbol{z}_{\geq \ell})$ as the rate accumulated in all layers from $\boldsymbol{z}_L$ to $\boldsymbol{z}_\ell$. The first inequality in Eq. 10 comes from the data processing inequality (MacKay,

2003), and the other two inequalities result from discarding the (nonnegative) KL-terms. The classification accuracy is thus bounded by (Meyen, 2016) (see also proof in Appendix B)

$$\text{class. accuracy} \leq f^{-1}\big(I_q(y; \boldsymbol{z}_\ell)\big) \leq f^{-1}\big(\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[R(\boldsymbol{z}_{\geq \ell})]\big) \quad \big(\leq f^{-1}\big(\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[R]\big)\big) \tag{11}$$

where f^{-1} is the inverse of the monotonic function $f(\alpha) = H[p_{\text{data}}(y)] + \alpha \log \alpha + (1-\alpha) \log \frac{1-\alpha}{M-1}$ with M being the number of classes and $H[p_{\text{data}}(y)] \leq \log M$ the marginal label entropy. Eq. 11 suggests that the accuracy of an optimal classifier on $\boldsymbol{z}_\ell$ would increase as the rate $R(\boldsymbol{z}_{\geq \ell})$ accumulated from $\boldsymbol{z}_L$ to $\boldsymbol{z}_\ell$ grows (i.e., as $\beta_{\geq \ell} \to 0$), and that the rate added in downstream layers $\boldsymbol{z}_{<\ell}$ would be irrelevant. Practical classifiers, however, have a limited expressiveness, which a very high rate $R(\boldsymbol{z}_{\geq \ell})$ might exceed by encoding too many details into $\boldsymbol{z}_\ell$ that are not necessary for classification. We observe in Section 5.6 that, in such cases, increasing the rates of *downstream* layers $\boldsymbol{z}_{<\ell}$ improves classification accuracy as it allows keeping $\boldsymbol{z}_\ell$ simpler by deferring details to $\boldsymbol{z}_{<\ell}$.

Data Generation. The original proposal of VAEs (Kingma & Welling, 2014) motivated them from a generative modeling perspective using that, for $\beta = 1$, the negative of the loss function in Eq. 2 is a lower bound on the log marginal data likelihood. This suggests setting all β-hyperparameters in Eq. 8 to values close to 1 if a HVAE is used primarily for its generative model p_θ.

In summary, our theoretical analysis suggests that optimally tuned layer-wise rates depend on whether a HVAE is used for data reconstruction, representation learning, or data generation. The next section tests our theoretical predictions empirically for the same three application domains.

5 Experiments

To demonstrate the features of our hierarchical information trading framework, we run large-scale grid searches over a two-dimensional rate space using two different implementations of HVAEs and three different data sets. Although the proposed framework is applicable for HVAEs with $L \geq 2$, we only use HVAEs with $L = 2$ in our experiments for simplicity and visualization purpose.

5.1 Experimental setup

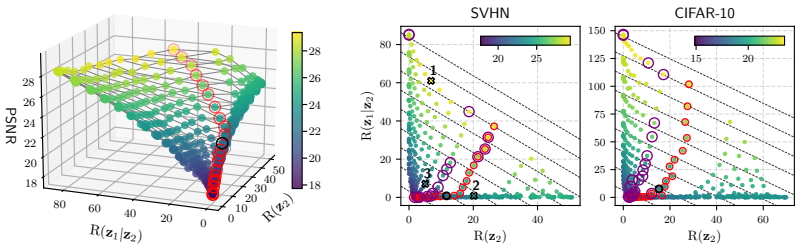
Data sets. We used the SVHN (Netzer et al., 2011) and CIFAR-10 (Krizhevsky, 2009) data sets (both 32×32 pixel color images), and MNIST (LeCun et al., 1998) (28×28 binary pixel images). SVHN consists of photographed house numbers from 0 to 9, which are geometrically simpler than the 10 classes of objects from CIFAR-10 but more complex than MNIST digits. Most results shown in the main paper use SVHN; comprehensive results for CIFAR-10 and MNIST are shown in Appendix A.2 and tell a similar story except where explicitly discussed.

Model Architectures. For the generative model (Eq. 1), we assume a (fixed) standard Gaussian prior $p(\boldsymbol{z}_2) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and we use diagonal Gaussian models for $p_\theta(\boldsymbol{z}_1|\boldsymbol{z}_2) = \mathcal{N}(g_\mu(\boldsymbol{z}_2), g_\sigma(\boldsymbol{z}_2)^2)$ and (for SVHN and CIFAR-10) $p_\theta(\boldsymbol{x}|\boldsymbol{z}_1) = \mathcal{N}(g_{\mu'}(\boldsymbol{z}_1), \sigma_{\boldsymbol{x}}^2 \mathbf{I})$ (this is similar to, e.g., (Minnen et al., 2018)). Here, g_μ, g_σ, and $g_{\mu'}$, denote neural networks (see details below). Since MNIST has binary pixel values, we model it with a Bernoulli distribution for $p_\theta(\boldsymbol{x}|\boldsymbol{z}_1) = \text{Bern}(g_{\mu'}(\boldsymbol{z}_1))$. For the inference model, we also use diagonal Gaussian models for $q_\phi(\boldsymbol{z}_2|\boldsymbol{x}) = \mathcal{N}(f_\mu(\boldsymbol{x}), f_\sigma(\boldsymbol{x})^2)$ and for $q_\phi(\boldsymbol{z}_1|\boldsymbol{x}, \boldsymbol{z}_2) = \mathcal{N}(f_{\mu'}(\boldsymbol{x}, \boldsymbol{z}_2), f_{\sigma'}(\boldsymbol{x}, \boldsymbol{z}_2)^2)$, where f_μ, f_σ, $f_{\mu'}$, and $f_{\sigma'}$ are again neural networks.

We examine both LVAE (Figure 2(b)) and our generalized top-down HVAEs (GHVAEs; see Figure 2(c)), using simple network architectures with only 2 to 3 convolutional and 1 fully connected layers (see Appendix A.1 for details) so that we can scan a large rate-space efficiently. Note that we are not trying to find the new state-of-the-art HVAEs. Results for LVAE are in Appendix A.2.2.

We trained 441 different HVAEs for each data set/model combination, scanning the rate-hyperparameters (β_2, β_1) over a 21×21 grid ranging from 0.1 to 10 on a log scale in both directions (see Figure 1 on page 2, right panels). Each model took about 2 hours to train on an RTX-2080Ti GPU ($\sim$27 hours in total for each data set/model combination using 32 GPUs in parallel).

Baselines. Our proposed framework (Eq. 8) generalizes over both VAEs and β-VAEs (Eq. 2), which we obtain in the cases $\beta_2 = \beta_1 = 1$ and $\beta_2 = \beta_1$, respectively. These baselines are indicated as black "o" and red "o" circles, respectively, in Figures 3, 5, 6, and 7, discussed below.



(a) Rate/rate/distortion surface for SVHN.

(b) PSNR-rates comparison in 2d.

Figure 3: PSNR-rate trade-off for GHVAEs trained on SVHN and CIFAR-10. Figure (a) visualizes the same data as the left panel of (b) in 3d. Black circles "○" mark standard VAEs ($\beta_2 = \beta_1 = 1$), red circles "○" mark β-VAEs ($\beta_2 = \beta_1$), and purple circles "○" mark optimal models along constant total rate (dashed diagonal lines) as defined in Section 5.3. Crosses point to columns in Figure 4.

Metrics. Performance metrics for the three application domains of VAEs mentioned in the introduction are introduced at the beginnings of the corresponding Sections 5.4-5.6. In addition, we evaluate the individual rates $R(\mathbf{z}_2)$ and $R(\mathbf{z}_1|\mathbf{z}_2)$ (Eq. 7), which we report in *nats* (i.e., to base e).

5.2 There is no "One HVAE Fits All"

Figure 1 on page 2 summarizes our results. The 21×21 GHVAEs trained with the grid of hyperparameters β_2 and β_1 map out a surface in a 3d-space spanned by suitable metrics for the three application domains (metrics defined in Sections 5.4-5.6 below). The two upper right panels map colors on this surface to βs used for training and to the resulting layer-wise rates, respectively. The lower right panels show performance landscapes and identify the optimal models for the three application domains of data reconstruction (△), representation learning (◇), and generative modeling (⬠).

The figure shows that moving away from a conventional β-VAE ($\beta_2 = \beta_1$; dashed red lines in Figure 1) allows us to find better models for a given application domain as the three application domains favor vastly different regions in β-space. Thus, *there is no single HVAE that is optimal for all tasks*, and a HVAE that has been optimized for one task can perform poorly on a different task.

5.3 Definition of the Optimal Model for a Given Total Rate

One of the questions we study in Sections 5.4-5.6 below is: "Which allocation of rates across layers results in best model performance *if we keep the total rate R fixed*". Unfortunately, it is difficult to keep R fixed at training time since we control rates only indirectly via their Lagrange multipliers β_2 and β_1. We instead use the following definition, illustrated in Figure 6 for a performance metric introduced in Section 5.6 below. The figure plots the performance metric over R for all 21×21 β-settings and highlights with purple circles "○" all points on the upper convex hull. These highlighted models are optimal for a small interval of total rates in the following sense: if we use the total rates R of all "○" to partition the horizontal axis into intervals then, by definition of the convex hull, each "○" represents the model with highest performance in either the interval to its left or the one to its right.

5.4 Performance on Data Reconstruction

Reconstruction is a popular task for VAEs, e.g., in the area of lossy compression (Ballé et al., 2017). We measure reconstruction quality using the common peak signal-to-noise ratio (PSNR), which is equal to $\mathbb{E}_{\boldsymbol{x} \sim \mathbb{X}_{\text{test}}}[-\log D]$ up to rescaling and shifting. Higher PSNR means better reconstruction.

Figure 3(a) shows a 3d-plot of PSNR as a function of both $R(\boldsymbol{z}_1|\boldsymbol{z}_2)$ and $R(\boldsymbol{z}_2)$ for SVHN, thus generalizing the rate/distortion curve of a conventional β-VAE to a rate/rate/distortion surface. Figure 3(b) introduces a more compact 2d-representation of the same data that we use for all remaining metrics in the rest of this section and in Appendix A.2, and it also shows results for CIFAR-10.

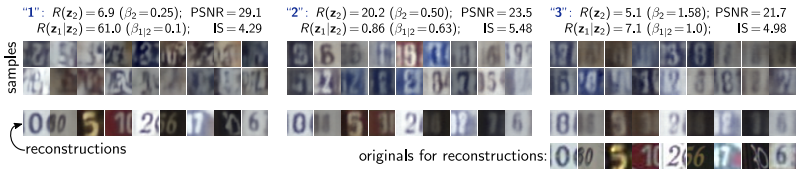


Figure 4: Samples (top) and reconstructions (bottom) from 3 different models (blue column labels "1", "2", and "3" from left to right correspond to crosses "1", "2", and "3" in Figures 3(b) & 5). Consistent with PSNR and IS metrics, model "1" produces poorest samples but best reconstructions.

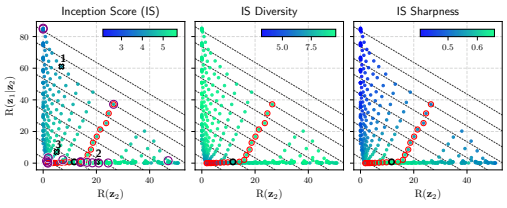


Figure 5: Sample generation performance, measured in Inception Score (IS, see Eq. 12) and its factorization into diversity and sharpness as a function of layer-wise rates for GHVAEs trained using SVHN data. Crosses in left panel correspond to samples shown in Figure 4. Markers "o", "o", and "o" same as in Figure 3.

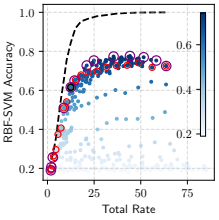


Figure 6: RBF-SVM classification accuracies on $\boldsymbol{\mu}_2$. Dashed line shows theoretical bound (Eq. 11). Other markers as in Figure 3.

Unsurprisingly and consistent with Eq. 9, reconstruction performance improves as total rate grows. However, minimizing distortion without any constraints is not useful in practice as we can simply use the original data, which has no distortion. To simulate a practical constraint in, e.g., a data-compression application, we consider models with optimal PSNR *for a given total rate* R (as defined in Section 5.3) which are marked as purple circles "o" in Figure 3(b). We see for both SVHN and CIFAR-10 that conventional β-VAEs ($\beta_2 = \beta_1$; red circles) perform somewhat suboptimal for a given total rate and can be improved by trading some rate in $\boldsymbol{z}_2$ for some rate in $\boldsymbol{z}_1$. Reconstruction examples for the three models marked with crosses in Figure 3(b) are shown in Figure 4 (bottom). Visual reconstruction quality improves from "3" to "2" to "1", consistent with reported PSNRs.

5.5 Performance on Sample Generation

We next evaluate how tuning layer-wise rates affects the quality of samples from the generative model. We measure sample quality by the widely used Inception Score (IS) (Salimans et al., 2016),

$$\mathrm{IS} = \exp\left\{\mathbb{E}_{p_\theta(\boldsymbol{x})}\left[D_{\mathrm{KL}}[p_{\mathrm{cls.}}(y|\boldsymbol{x}) \,\|\, p_{\mathrm{cls.}}(y)]\right]\right\} = e^{H[p_{\mathrm{cls.}}(y)]} \times e^{-\mathbb{E}_{p_\theta(\boldsymbol{x})}[H[p_{\mathrm{cls.}}(y|\boldsymbol{x})]]} \tag{12}$$

Here, p_θ is the trained generative model (Eq. 1), $p_{\mathrm{cls.}}(y|\boldsymbol{x})$ is the predictive distribution of a classifier trained on the same training set, and $p_{\mathrm{cls.}}(y) := \mathbb{E}_{p_\theta(\boldsymbol{x})}[p_{\mathrm{cls.}}(y|\boldsymbol{x})]$. The second equality in Eq. 12 follows Barratt & Sharma (2018) to split IS into a product of a diversity score and a sharpness score. Higher is better for all scores. The classifier is a ResNet-18 (He et al., 2016) for SVHN (test accuracy $95.02\,\%$) and a DenseNet-121 (Huang et al., 2017) for CIFAR-10 (test accuracy $94.34\,\%$).

Figure 5 (left) shows IS for GHVAEs trained on SVHN. Unlike the results for PSNR, here, higher rate does not always lead to better sample quality: for very high $R(\mathbf{z}_2)$ and low $R(\mathbf{z}_1|\mathbf{z}_2)$, IS eventually drops. The region of high IS is in the area where $\beta_2 < \beta_1$, i.e., where $R(\mathbf{z}_2)$ is higher than in a comparable conventional β-VAE. The center and right panels of Figure 5 show diversity and sharpness, indicating that IS is mainly driven here by sharpness, which depends mostly on $R(\mathbf{z}_2)$, possibly because $\boldsymbol{z}_2$ captures higher-level concepts than $\boldsymbol{z}_1$ that may be more important to the classifier in Eq. 12. Samples from the the three models marked with crosses in Figure 5 are shown in Figure 4 (top). Visual sample quality improves from "1" to "3" to "2", consistent with reported IS.

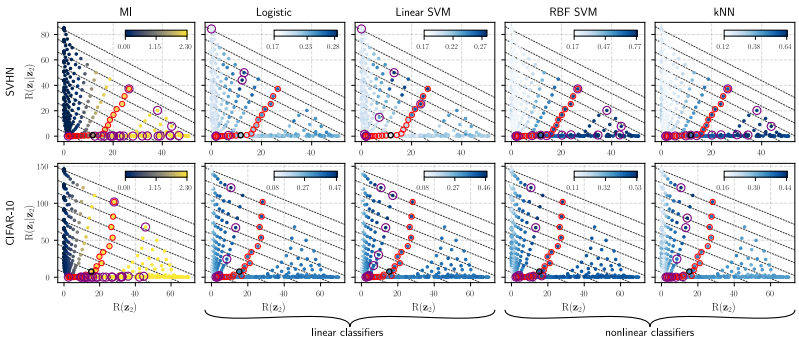


Figure 7: Mutual information (MI) $I_q(y;\boldsymbol{z}_2)$ and classification accuracies of four classifiers (see column labels) as a function of layer-wise rates $R(\boldsymbol{z}_2)$ and $R(\boldsymbol{z}_1|\boldsymbol{z}_2)$. Classifiers are conditioned on $\boldsymbol{\mu}_2 := \arg\max_{\boldsymbol{z}_2} q(\boldsymbol{z}_2|\boldsymbol{x})$ learned from GHVAEs trained with SVHN (top) and CIFAR-10 (bottom). Markers "o", "o", and "o" same as in Figure 3.

5.6 Performance on Representation Learning for Downstream Classification

VAEs are very popular for representation learning as they map complicated high dimensional data $\boldsymbol{x}$ to typically lower dimensional representations $\{\boldsymbol{z}_\ell\}$. To measure the quality of learned representations, we train two sets of classifiers on a labeled test set for each trained HVAE, each consisting of: logistic regression, a Support Vector Machine (SVM) (Boser et al., 1992) with linear kernel, an SVM with RBF kernel, and k-nearest neighbors (kNN) with $k = 5$. One set of classifiers is conditioned on the mode $\boldsymbol{\mu}_2$ of $q_\phi(\boldsymbol{z}_2|\boldsymbol{x})$ and the other one on the mode $\boldsymbol{\mu}_1$ of $q_\phi(\boldsymbol{z}_1|\boldsymbol{z}_2,\boldsymbol{x})$, where $\boldsymbol{z}_2 \sim q_\phi(\boldsymbol{z}_2|\boldsymbol{x})$. We use the implementations from scikit-learn (Pedregosa et al., 2011) for all classifiers.

Figure 7 shows the classification accuracies (columns 2-5) for all classifiers trained on $\boldsymbol{\mu}_2$. The first column shows the mutual information $I_q(y;\boldsymbol{z}_2)$, which depends mainly on $R(\boldsymbol{z}_2)$ as expected from Eq. 10. As long as the classifier is expressive enough (e.g., RBF-SVM or kNN) and the data set is simple (SVHN; top row), higher mutual information ($\approx$ higher $R(\boldsymbol{z}_2)$) corresponds to higher classification accuracies, consistent with Eq. 11. But for less expressive (e.g., linear) classifiers or more complex data (CIFAR-10; bottom row), increasing $R(\boldsymbol{z}_1|\boldsymbol{z}_2)$ improves classification accuracy (see purple circles "o" in corresponding panels), consistent with the discussion below Eq. 11. We see a similar effect (Table 1) for most classifier/data set combinations when replacing $\boldsymbol{\mu}_2$ by $\boldsymbol{\mu}_1$, which has more information about $\boldsymbol{x}$ but is also higher dimensional.

Table 1: Optimal classification accuracies (across all (β_2,β_1)-settings) using either $\boldsymbol{\mu}_2$ or $\boldsymbol{\mu}_1$.

Data Set	log. reg.	lin. SVM	RBF SVM	kNN
SVHN ($\boldsymbol{\mu}_2$)	28.43 %	27.87 %	**77.60 %**	**64.25 %**
SVHN ($\boldsymbol{\mu}_1$)	**45.77 %**	**49.81 %**	59.28 %	56.49 %
CIFAR-10 ($\boldsymbol{\mu}_2$)	**47.36 %**	**46.95 %**	**53.15 %**	**44.20 %**
CIFAR-10 ($\boldsymbol{\mu}_1$)	43.27 %	42.55 %	45.60 %	39.25 %

6 Conclusions

We classified the various tasks that can be performed with Variational Autoencoders (VAEs) into three application domains and argued that each domain has different trade-offs, such that a good VAE for one domain is not necessarily good for another. This observation motivated us to propose a refinement of the rate/distortion theory of VAEs that allows trading off rates across individual layers of latents in hierarchical VAEs. We showed both theoretically and empirically that the proposal indeed provides practitioners better control for tuning VAEs for the three application domains. In the future, it would be interesting to explore adaptive schedules for the Lagrange parameters $\boldsymbol{\beta}$ that would make it possible to target a specific given rate for each layer in a single training run, for example by using the method proposed by Rezende & Viola (2018).

ACKNOWLEDGMENTS

The authors would like to thank Johannes Zenn, Zicong Fan, Zhen Liu for their helpful discussion. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Tim Z. Xiao.

Reproducibility Statement. All code necessary to reproduce the results in this paper is available at https://github.com/timxzz/HIT/.

REFERENCES

Eirikur Agustsson and Lucas Theis. Universally quantized neural compression. *Advances in Neural Information Processing Systems*, 33:12367–12376, 2020. 4

Alexander Alemi, Ben Poole, Ian Fischer, Joshua Dillon, Rif A Saurous, and Kevin Murphy. Fixing a broken elbo. In *International Conference on Machine Learning*, pp. 159–168. PMLR, 2018. 1, 2, 3, 4

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations*, 2017. 1, 3, 4

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations*, 2017. 1, 3, 7

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018. 2

Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018. 8

Charles H Bennett, Peter W Shor, John A Smolin, and Ashish V Thapliyal. Entanglement-assisted capacity of a quantum channel and the reverse shannon theorem. *IEEE transactions on Information Theory*, 48(10):2637–2655, 2002. 4

Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992. 9

Rewon Child. Very deep vaes generalize autoregressive models and can outperform them on images. In *International Conference on Learning Representations*, 2021. 3

Zhiwei Deng, Rajitha Navarathna, Peter Carr, Stephan Mandt, Yisong Yue, Iain Matthews, and Greg Mori. Factorized variational autoencoders for modeling audience reactions to movies. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2577–2586, 2017. 2

Brendan J Frey and GE Hinton. E cient stochastic source coding and an application to a bayesian network source model. *Computer Journal*, 1997. 4

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. In *International Conference on Learning Representations*, 2017. 3

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 8

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. 1, 3, 4

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017. 8

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: an unsupervised and generative approach to clustering. In *International Joint Conference on Artificial Intelligence*, 2017. 1

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014. 1, 3, 4, 6

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 6

Eric Laloy, Romain Hérault, John Lee, Diederik Jacques, and Niklas Linde. Inversion using a new low-dimensional representation of complex binary geological media based on a deep neural network. *Advances in water resources*, 110:387–405, 2017. 1

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. Biva: A very deep hierarchy of latent variables for generative modeling. *Advances in Neural Information Processing Systems*, 32, 2019. 3

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. 5

Sascha Meyen. Relation between classification accuracy and mutual information in equally weighted classification tasks. Master's thesis, Universität Hamburg, 2016. 5, 6, 17

David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in Neural Information Processing Systems*, 31, 2018. 3, 6

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 6

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 9

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in Neural Information Processing Systems*, 32, 2019. 1

Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *arXiv preprint arXiv:1810.00597*, 2018. 9

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014. 1, 3

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in Neural Information Processing Systems*, 29, 2016. 8

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. *Advances in Neural Information Processing Systems*, 29, 2016. 3, 4

Hiroshi Takahashi, Tomoharu Iwata, Yuki Yamanaka, Masanori Yamada, and Satoshi Yagi. Student-t variational autoencoder for robust density estimation. In *International Joint Conference on Artificial Intelligence*, 2018. 1

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*, pp. 1–5. IEEE, 2015. 1, 3

Arash Vahdat and Jan Kautz. NVAE: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, 33:19667–19679, 2020. 3

Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 world wide web conference*, pp. 187–196, 2018. 1

Yibo Yang, Robert Bamler, and Stephan Mandt. Improving inference for neural image compression. *Advances in Neural Information Processing Systems*, 33:573–584, 2020. 3

Li Yingzhen and Stephan Mandt. Disentangled sequential autoencoder. In *International Conference on Machine Learning*, 2018. 2

A EXPERIMENT SUPPLEMENTARIES

A.1 IMPLEMENTATION DETAILS

Table 2: Model architecture details for generalized top-down HVAEs (GHVAEs) used in Section 5. *Conv* and *ConvTransp* denote the convolutional and transposed convolutional layer, which has the corresponding input: *input channel, output channel, kernel size, stride, padding*. *FC* represents fully connected layer.

Data set	$q(\boldsymbol{z}_2\|\boldsymbol{x})$	$q(\boldsymbol{z}_1\|\boldsymbol{z}_2,\boldsymbol{x})$	$p(\boldsymbol{z}_1\|\boldsymbol{z}2)$	$p(\boldsymbol{x}\|\boldsymbol{z}_1)$
SVHN/ CIFAR-10	*Share:* Conv(3, 32, 4, 2, 1), Conv(32, 32, 4, 2, 1), Conv(32, 32, 4, 2, 1) *For mean:* FC(In=512, Out=32) *For variance:* FC(In=512, Out=32)	*For* $\boldsymbol{x}$*:* Conv(3, 32, 4, 2, 1), Conv(32, 32, 4, 2, 1) *For* $\boldsymbol{z}_2$*:* FC(In=32, Out=512) *Share:* ConvTransp(64, 32, 4, 2, 1) *For mean:* Conv(32, 32, 3, 1, 1) *For variance:* Conv(32, 32, 3, 1, 1)	*Share:* FC(In=32, Out=256) *For mean:* FC(In=256, Out=512) *For variance:* FC(In=256, Out=512)	ConvTransp(32, 32, 4, 2, 1), ConvTransp(32, 32, 4, 2, 1), ConvTransp(32, 3, 4, 2, 1)
	$\boldsymbol{z}_1$ dims: 512	$\boldsymbol{z}_2$ dims: 32	$\sigma_{\boldsymbol{x}} = 0.71$	Total params: 475811
MNIST (Binary)	*Share:* Conv(1, 16, 4, 2, 1), Conv(16, 16, 4, 2, 1), Conv(16, 16, 4, 1, 0) *For mean:* FC(In=256, Out=20) *For variance:* FC(In=256, Out=20)	*For* $\boldsymbol{x}$*:* Conv(1, 16, 4, 2, 1), Conv(16, 16, 4, 1, 0) *For* $\boldsymbol{z}_2$*:* FC(In=20, Out=256) *Share:* ConvTransp(32, 16, 4, 1, 0) *For mean:* Conv(16, 16, 3, 1, 1) *For variance:* Conv(16, 16, 3, 1, 1)	*Share:* FC(In=20, Out=128) *For mean:* FC(In=128, Out=256) *For variance:* FC(In=128, Out=256)	ConvTransp(16, 16, 4, 1, 0), ConvTransp(16, 16, 4, 2, 1), ConvTransp(16, 1, 4, 2, 1)
	$\boldsymbol{z}_1$ dims: 256	$\boldsymbol{z}_2$ dims: 20	$\sigma_{\boldsymbol{x}}$: N/A	Total params: 122713

Table 3: Model architecture details for LVAEs used in Section 5. *Conv* and *ConvTransp* denote the convolutional and transposed convolutional layer, which has the corresponding input: *input channel, output channel, kernel size, stride, padding*. *FC* represents fully connected layer.

Data set	$q(\boldsymbol{z}_2\|\boldsymbol{x})$	$q(\boldsymbol{z}_1\|\boldsymbol{z}_2,\boldsymbol{x})$	$p(\boldsymbol{z}_1\|\boldsymbol{z}2)$	$p(\boldsymbol{x}\|\boldsymbol{z}_1)$
SVHN/ CIFAR-10	*Share:* Conv(3, 32, 4, 2, 1), Conv(32, 32, 4, 2, 1), Conv(32, 32, 4, 2, 1) *For mean:* FC(In=512, Out=32) *For variance:* FC(In=512, Out=32)	*Involve* $\mathbf{d}$*:* Conv(32, 32, 4, 2, 1) *For mean:* Conv(32, 32, 3, 1, 1) *For variance:* Conv(32, 32, 3, 1, 1)	*Share:* FC(In=32, Out=256) *For mean:* FC(In=256, Out=512) *For variance:* FC(In=256, Out=512)	ConvTransp(32, 32, 4, 2, 1), ConvTransp(32, 32, 4, 2, 1), ConvTransp(32, 3, 4, 2, 1)
	$\boldsymbol{z}_1$ dims: 512	$\boldsymbol{z}_2$ dims: 32	$\sigma_{\boldsymbol{x}} = 0.71$	Total params: 408131

A.2 ADDITIONAL RESULTS

Here we attached the results for MNIST, as well as the full results for LVAE on SVHN and generalized top-down HVAEs on CIFAR-10.

A.2.1 RESULTS FOR GENERALIZED TOP-DOWN HVAES ON MNIST

We also evaluate our proposed framework using generalized top-down HVAEs trained on binary MNIST data (i.e., black and white images rather than grayscale).

We note that the inception score (IS) behaves different in our MNIST models compared to SVHN (see Figure 5) in that optimal IS in MNIST occurs for high $R(\boldsymbol{z}_1|\boldsymbol{z}_2)$ rather than high $R(\boldsymbol{z}_2)$. This indicates that semantically low-level properties (hand-writing style) of MNIST might have more variation than high level properties (the digit), whereas SVHN images show variation in additional high-level properties such as the background color.

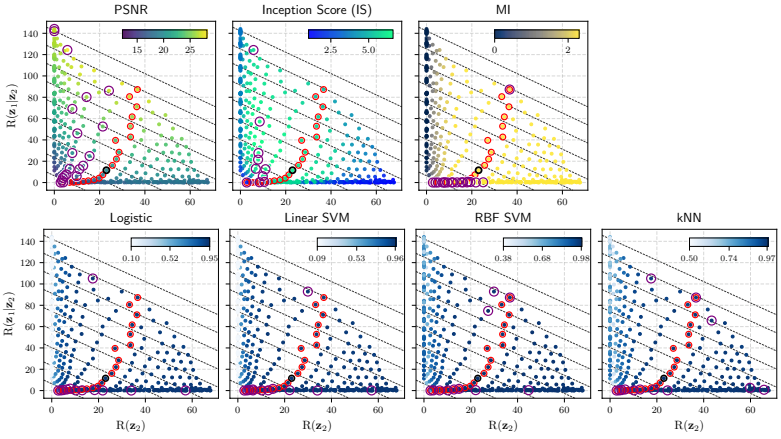


Figure 8: Trade-offs between rates and all metrics we used in Section 5 from the generalized top-down HVAEs trained with MNIST. The results from the standard VAE (i.e. $\beta_2 = \beta_1 = 1$) and the β-VAE (i.e. $\beta_2 = \beta_1$) are marked with "o" and "o". The markers "o" highlight the optimal models selected using convex hull (see Figure 6 for details). The diagonal grid lines are references for equivalent total rates, i.e. points on the same line have the same total rates.

A.2.2 RESULTS FOR LVAE ON SVHN

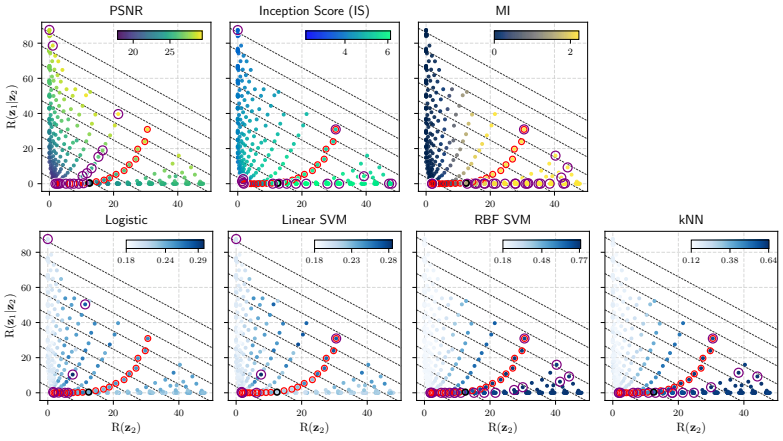


Figure 9: Trade-offs between rates and all metrics we used in Section 5 from LVAE trained with SVHN. The results from the standard VAE (i.e. $\beta_2 = \beta_1 = 1$) and the β-VAE (i.e. $\beta_2 = \beta_1$) are marked with "o" and "o". The markers "o" highlight the optimal models selected using convex hull (see Figure 6 for details). The diagonal grid lines are references for equivalent total rates, i.e. points on the same line have the same total rates.

A.2.3 RESULTS FOR GENERALIZED TOP-DOWN HVAES ON CIFAR-10

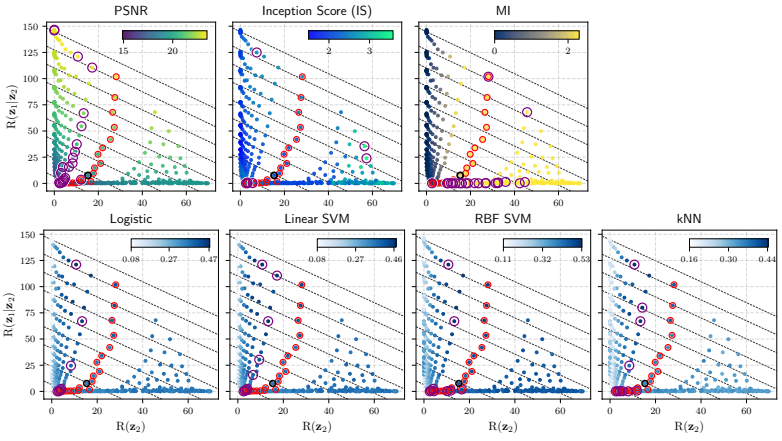


Figure 10: Trade-offs between rates and all metrics we used in Section 5 from the generalized top-down HVAEs trained with CIFAR-10. The results from the standard VAE (i.e. $\beta_2 = \beta_1 = 1$) and the β-VAE (i.e. $\beta_2 = \beta_1$) are marked with "o" and "o". The markers "o" highlight the optimal models selected using convex hull (see Figure 6 for details). The diagonal grid lines are references for equivalent total rates, i.e. points on the same line have the same total rates.

B PROOF OF THE BOUND ON CLASSIFICATION ACCURACY

This section provides a proof of Eq. 11 by reformulating the proof of Proposition 5 in the thesis by Meyen (2016) into the notation used in the present paper. We stress that this section contains no original contribution and is provided only as a convenience to the reader, motivated by reviewer feedback. All credits for this section belong to Meyen (2016).

We consider an (unknown) true data generative distribution $p_{\text{data}}(y, \boldsymbol{x})$ for data $\boldsymbol{x}$ with (unobserved) true labels y, and a hierarchical VAE with an inference model $q_\phi(\{\boldsymbol{z}_\ell\} \,|\, \boldsymbol{x})$ of the form of Eq. 4. Focusing on a single layer ℓ of latents, we denote the joint probability over y, $\boldsymbol{x}$, and $\boldsymbol{z}_\ell$ as

$$q(y, \boldsymbol{x}, \boldsymbol{z}_\ell) := p_{\text{data}}(y, \boldsymbol{x})\, q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x}) \tag{13}$$

where the marginal $q_\phi(\boldsymbol{z}_\ell|\boldsymbol{x})$ of $q_\phi(\{\boldsymbol{z}_\ell\} \,|\, \boldsymbol{x})$ is defined as usual. We further consider a classifier $p_{\text{cls.}}(y|\boldsymbol{z}_\ell)$ that operates on $\boldsymbol{z}_\ell$. Denoting its top prediction as $\hat{y} := \arg\max_y p_{\text{cls.}}(y|\boldsymbol{z}_\ell)$, the classification accuracy is $\alpha := \mathbb{E}_q[\delta_{y,\hat{y}}]$, where δ is the Kronecker delta.

Theorem 1. *The mutual information $I_q(y; \boldsymbol{z}_\ell)$ between the latent representation $\boldsymbol{z}_\ell$ and the true label y under the distribution q defined in Eq. 13 is lower bounded as follows,*

$$I_q(y; \boldsymbol{z}_\ell) \geq f(\alpha) \qquad \textit{with} \qquad f(\alpha) = H_{p_{data}}[y] - H_2(\alpha) - (1-\alpha)\log(M-1) \tag{14}$$

where $H_2(\alpha) = -\alpha \log \alpha - (1-\alpha)\log(1-\alpha)$ is the entropy of a Bernoulli distribution, $H_{p_{data}}[y] \leq \log M$ is the marginal entropy of the true labels, and M denotes the number of classes.

Before we prove Theorem 1, we note that the function f is strictly monotonically increasing on the relevant interval $[\max_y p_{\text{data}}(y), 1]$. Thus, f is invertible and we obtain the following corollary:

Corollary 1. *The classification accuracy α is upper bounded as in Eq. 11 of the main text, i.e.,*

$$\alpha \leq f^{-1}(I_q(y; \boldsymbol{z}_\ell)) \leq f^{-1}\big(\mathbb{E}_{p_{data}(\boldsymbol{x})}[R(\boldsymbol{z}_{\geq \ell})]\big). \tag{15}$$

The second inequality in Eq. 15 results from the bound $I_q(y; \boldsymbol{z}_\ell) \leq \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[R(\boldsymbol{z}_{\geq \ell})]$ derived in Eq. 10, using the fact that f^{-1} is monotonically increasing (since f is).

Proof of Theorem 1. We split the mutual information into two contributions,

$$I_q(y; \boldsymbol{z}_\ell) = H_{p_{\text{data}}}[y] - H_q[y|\boldsymbol{z}_\ell] = H_{p_{\text{data}}}[y] - \mathbb{E}_{\boldsymbol{z}_\ell \sim q(\boldsymbol{z}_\ell)}\big[\mathbb{E}_{y\sim q(y|\boldsymbol{z}_\ell)}[-\log q(y|\boldsymbol{z}_\ell)]\big] \tag{16}$$

where, as clarified in the second equality, $H_q[y|\boldsymbol{z}_\ell]$ is the expectation over $\boldsymbol{z}_\ell$ of the conditional entropy of y given $\boldsymbol{z}_\ell$, and $q(\boldsymbol{z}_\ell)$ and $q(y|\boldsymbol{z}_\ell)$ are marginals and conditionals of q (Eq. 13) as usual.

Since $H_{p_{\text{data}}}[y]$ is fixed by the problem at hand, finding a lower bound on $I_q(y; \boldsymbol{z}_\ell)$ for a given classification accuracy α is equivalent to finding an upper bound on the second term on the right-hand side of Eq. 16, $H_q[y|\boldsymbol{z}_\ell] = \mathbb{E}_{\boldsymbol{z}_\ell \sim q(\boldsymbol{z}_\ell)}[\mathbb{E}_{y\sim q(y|\boldsymbol{z}_\ell)}[-\log q(y|\boldsymbol{z}_\ell)]]$, with the constraint $\mathbb{E}_q[\delta_{y,\hat{y}}] = \alpha$. We do this by upper bounding the conditional entropy $\mathbb{E}_{y\sim q(y|\boldsymbol{z}_\ell)}[-\log q(y|\boldsymbol{z}_\ell)]$ of y given $\boldsymbol{z}_\ell$ for all $\boldsymbol{z}_\ell$ independently, and then taking the expectation over $\boldsymbol{z}_\ell \sim q(\boldsymbol{z}_\ell)$.

For a fixed latent representation $\boldsymbol{z}_\ell$, we first split off the contribution to $\mathbb{E}_{y\sim q(y|\boldsymbol{z}_\ell)}[-\log q(y|\boldsymbol{z}_\ell)]$ from $y = \hat{y}$, where $\hat{y} = \arg\max_y p_{\text{cls.}}(y|\boldsymbol{z}_\ell)$ is the label that our classifier would predict for $\boldsymbol{z}_\ell$,

$$\mathbb{E}_{y\sim q(y|\boldsymbol{z}_\ell)}[-\log q(y|\boldsymbol{z}_\ell)] = -q(y{=}\hat{y}|\boldsymbol{z}_\ell)\log q(y{=}\hat{y}|\boldsymbol{z}_\ell) - \sum_{y\neq\hat{y}} q(y|\boldsymbol{z}_\ell)\log q(y|\boldsymbol{z}_\ell). \tag{17}$$

Here, the second term on the right-hand side resembles the entropy of a distribution over the remaining $(M-1)$ labels ($y \neq \hat{y}$), except that the probabilities sum to $(1 - q(y{=}\hat{y}|\boldsymbol{z}_\ell))$ rather than one. Thus, regardless of the value of $q(y{=}\hat{y}|\boldsymbol{z}_\ell)$, this term is maximized if $q(y|\boldsymbol{z}_\ell)$ distributes the remaining probability mass $(1 - q(y{=}\hat{y}|\boldsymbol{z}_\ell))$ uniformly over the remaining $(M-1)$ labels, i.e.,

$$\begin{aligned}\mathbb{E}_{y\sim q(y|\boldsymbol{z}_\ell)}[-\log q(y|\boldsymbol{z}_\ell)] &\leq -q(y{=}\hat{y}|\boldsymbol{z}_\ell)\log q(y{=}\hat{y}|\boldsymbol{z}_\ell) - (1-q(y{=}\hat{y}|\boldsymbol{z}_\ell))\log\frac{1-q(y{=}\hat{y}|\boldsymbol{z}_\ell)}{M-1} \\ &= H_2(q(y{=}\hat{y}|\boldsymbol{z}_\ell)) + (1-q(y{=}\hat{y}|\boldsymbol{z}_\ell))\log(M-1).\end{aligned} \tag{18}$$

Plugging Eq. 18 back into Eq. 16, we obtain the bound

$$I_q(y; \boldsymbol{z}_\ell) \geq H_{p_{\text{data}}}[y] - \mathbb{E}_{\boldsymbol{z}_\ell\sim q(\boldsymbol{z}_\ell)}\big[H_2(q(y{=}\hat{y}|\boldsymbol{z}_\ell))\big] - \mathbb{E}_{\boldsymbol{z}_\ell\sim q(\boldsymbol{z}_\ell)}\big[1-q(y{=}\hat{y}|\boldsymbol{z}_\ell)\big]\log(M-1). \tag{19}$$

We arrive at the proposition (Eq. 14) by pulling the concave function H_2 out of the expectation using Jensen's inequality, and by then identifying $\mathbb{E}_{\boldsymbol{z}_\ell\sim q(\boldsymbol{z}_\ell)}[q(y{=}\hat{y}|\boldsymbol{z}_\ell)] = q(y{=}\hat{y}) = \alpha$. □

Published in Transactions on Machine Learning Research (12/2024)

A Note on Generalization in Variational Autoencoders: How Effective Is Synthetic Data & Overparameterization?

Tim Z. Xiao* *zhenzhong.xiao@uni-tuebingen.de*
AI Center Tübingen, University of Tübingen, IMPRS-IS

Johannes Zenn* *johannes.zenn@uni-tuebingen.de*
AI Center Tübingen, University of Tübingen, IMPRS-IS

Robert Bamler *robert.bamler@uni-tuebingen.de*
AI Center Tübingen, University of Tübingen

Reviewed on OpenReview: *https://openreview.net/forum?id=bwyHf5eery*

Abstract

Variational autoencoders (VAEs) are deep probabilistic models that are used in scientific applications (Cerri et al., 2019; Flam-Shepherd et al., 2022; Zhou & Wei, 2020; Gondur et al., 2023) and are integral to compression (Ballé et al., 2018; Minnen et al., 2018; Cheng et al., 2020; Yang et al., 2023); yet they are susceptible to overfitting (Cremer et al., 2018). Many works try to mitigate this problem from the *probabilistic methods perspective* by new inference techniques or training procedures. In this paper, we approach the problem instead from the *deep learning perspective* by investigating the effectiveness of using synthetic data and overparameterization for improving the generalization performance. Our motivation comes from (1) the recent discussion on whether the increasing amount of publicly accessible synthetic data will improve or hurt currently trained generative models (Alemohammad et al., 2024; Bohacek & Farid, 2023; Bertrand et al., 2024; Shumailov et al., 2023); and (2) the modern deep learning insights that overparameterization improves generalization. Our investigation shows how both training on samples from a pre-trained diffusion model, and using more parameters at certain layers are able to effectively mitigate overfitting in VAEs, therefore improving their generalization, amortized inference, and robustness performance. Our study provides timely insights in the current era of synthetic data and scaling laws.

1 Introduction

Variational autoencoders (VAEs, (Kingma & Welling, 2014; Rezende et al., 2014)) are deep probabilistic models that consist of an encoder and a decoder. VAEs were originally proposed in the framework of generative models, where the decoder (together with a prior) models the underlying data distribution $p_{\text{data}}(\boldsymbol{x})$, while the encoder plays only an auxiliary role to speed up training. Nowadays, however, many applications use VAEs for their encoder: VAEs are widely used in science, e.g., for anomaly detection (Cerri et al., 2019) and for learning interpretable representations in quantum optics (Flam-Shepherd et al., 2022) and neuroscience (Zhou & Wei, 2020; Gondur et al., 2023). In compression, VAEs are also the dominating models due to their connection to the rate-distortion theory (Ballé et al., 2018; Minnen et al., 2018; Cheng et al., 2020; Yang et al., 2023). These applications can be compromised if the VAE overfits. More specifically, it is empirically observed that the encoder $f_\phi(\boldsymbol{x})$ is more susceptible to overfitting than the decoder (Wu et al., 2017; Cremer et al., 2018; Shu et al., 2018). One possible explanation is that a VAE is normally trained with a finite dataset, due to the ELBO objective and the training algorithm, the encoder is fed with the same data at each epoch (by contrast, the decoder is fed with random samples from the approximate posterior distribution, therefore less likely to see the same data twice during training).

*Equal contribution.

Overfitting in the encoder implies that the learned mapping $f_\phi(\boldsymbol{x})$ does not generalize well to *unseen* data, which can negatively impact the performance of generative modeling, amortized inference, and adversarial robustness. For generative modeling, as the number of training epochs increases, an overfitted VAE will have a higher evidence lower bound (ELBO) on the training set but a lower ELBO on the test set. For amortized inference, an overfitted encoder is likely to map unseen data to a suboptimal set of variational parameters. This results in a lower ELBO when compared to the ELBO obtained by directly optimizing these parameters. For robustness, an overfitted encoder often learns a less smooth $f_\phi(\boldsymbol{x})$, i.e., a small change in $\boldsymbol{x}$ can result in a large difference in the latent space. This makes VAEs vulnerable to adversarial attacks, causing realistic and hardly distinguishable inputs to yield semantically different outputs (Kuzina et al., 2022).

In learning theory, overfitting is a result of the interaction between the size of a training set and the size of a model. According to the classical understanding, given a finite training set, as the model size (and presumably also the *model complexity*) increases, its performance first increases and then decreases, resulting in a U-shape relation between test error and model size. Conversely, the modern deep learning literature shows that if we keep increasing the model size, its performance gets better again, thus exhibiting a *double descent* behavior (Belkin et al., 2019; Nakkiran et al., 2021). In this work, we investigate to what extent these findings apply to VAEs, and whether we can mitigate overfitting in VAEs and improve their generalization performance by exploiting these findings. Starting from a VAE in the overfitting regime, we investigate the effect of increasing either the size of the training set or the size of the model. Since the generalization property that we investigate is mostly affected by the *relative* size of the model compared to the dataset, **we limit the discussion in this paper to datasets that** allow us to test strongly overparameterized models (compared to the dataset size), as well as models that are underparameterized.

There is a subtlety in increasing the training data for VAEs: Since the goal of VAEs is to model $p_{\text{data}}(\boldsymbol{x})$, naive data augmentations might alter the target distribution if not carefully crafted. Hence, we ask the question: "*Can we have infinite training samples drawn from* $p_{\text{data}}(\boldsymbol{x})$*?*" The answer is likely to be "No", unless we have access to the true data generating process. However, there is a class of models, known as diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021), that can estimate $p_{\text{data}}(\boldsymbol{x})$ very well, and that can generate as many samples as we want. Contrary to diffusion models, VAEs *learn* an encoding process leading to the so-called variational information bottleneck (Alemi et al., 2016). This incentivizes VAEs to encode information into their latents efficiently by removing redundancies from the input—different from diffusion models—which makes VAEs the most important tool in compression (Ballé et al., 2018; Minnen et al., 2018; Cheng et al., 2020; Yang et al., 2023). Additionally, the learned representations are fast, controllable, and interpretable which enables semantically meaningful representations useful for applications in e.g. science (Cerri et al., 2019; Flam-Shepherd et al., 2022; Zhou & Wei, 2020; Gondur et al., 2023). We therefore investigate whether training a VAE on samples from a pre-trained diffusion model alleviates overfitting in the encoder of the VAE, thus allowing us to learn better representations. This approach can be considered as *cross-model-class distillation*, i.e., distilling from a diffusion model to a VAE.

Contributions. We investigate generalization performance in VAEs from a deep learning perspective by considering synthetic data and overparameterization. Particularly, we show that in the overfitting regime (1) VAEs trained on samples from pre-trained diffusion models have better generalization, amortized inference and robustness performance; (2) we do not need infinite synthetic data to gain such generalization performance; and (3) increasing the number of parameters in the layers next to the latent variable (thus increasing the latent dimension) improves the generalization performance, but increasing the number of parameters in other layers hurts performance. Our findings provide practitioners with practical guidance to avoid overfitting in VAEs. Moreover, we find indications of the double descent phenomenon in VAEs which might be explained as for non-deep models (Curth et al., 2023).

2 Performance Metrics for VAEs

Variational autoencoders (VAEs). A VAE models the data distribution $p_{\text{data}}(\boldsymbol{x})$ by assuming a generative process that first draws a latent variable $\boldsymbol{z}$ from a prior $p_{\boldsymbol{z}}(\boldsymbol{z})$ and then draws $\boldsymbol{x}$ from a conditional likelihood $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$, parameterized by the output of a neural network ("decoder") $g_\theta(\boldsymbol{z})$ with weights θ. Given a training data distribution $p(\boldsymbol{x})$ approximating $p_{\text{data}}(\boldsymbol{x})$, naive maximum likelihood learning would

maximize the negative cross entropy $-H(p(\boldsymbol{x}), p_\theta(\boldsymbol{x})) = \mathbb{E}_{p(\boldsymbol{x})}[\log p_\theta(\boldsymbol{x})]$, where $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p_{\boldsymbol{z}}(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}$. However, as the integration over $\boldsymbol{z}$ is typically intractable, VAEs resort to variational inference (Blei et al., 2017) and instead maximize the evidence lower bound (ELBO)

$$\mathcal{L}(\Theta) = \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})}\big[\mathrm{ELBO}_\Theta(\boldsymbol{x})\big] \leq -H(p(\boldsymbol{x}), p_\theta(\boldsymbol{x})), \quad \text{where} \tag{1}$$

$$\mathrm{ELBO}_\Theta(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) + \log p_{\boldsymbol{z}}(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\big]. \tag{2}$$

Here, the variational distribution $q_\phi(\boldsymbol{z})$ is a simple probability distribution (often a fully factorized Gaussian) whose parameters are the outputs of the inference network (or "encoder") $f_\phi(\boldsymbol{x})$, whose weights ϕ are learned by maximizing $\mathcal{L}(\Theta)$ jointly over all parameters $\Theta = (\theta, \phi)$.

While we would like to use $p_{\mathrm{data}}(\boldsymbol{x})$ for $p(\boldsymbol{x})$, we only have access to a finite training set $\mathcal{D}_{\mathrm{train}}$, which can lead to overfitting. We discuss three performance gaps quantifying the effects of overfitting.

Generalization gap. One signal for overfitting is that a model performs better on the training set $\mathcal{D}_{\mathrm{train}}$ than on the test set $\mathcal{D}_{\mathrm{test}}$, and the test set performance decreases over training epochs. We refer to the difference between training and test set ELBO as the *generalization gap*,

$$\mathcal{G}_{\mathrm{g}} = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\mathrm{train}}}\left[\mathrm{ELBO}_\Theta(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\mathrm{test}}}\left[\mathrm{ELBO}_\Theta(\boldsymbol{x})\right]. \tag{3}$$

Since $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$ both consist of samples from the same distribution $p_{\mathrm{data}}(\boldsymbol{x})$, and training maximizes the ELBO on $\mathcal{D}_{\mathrm{train}}$, the ELBO on $\mathcal{D}_{\mathrm{train}}$ is typically greater than or equal to the ELBO on $\mathcal{D}_{\mathrm{test}}$, and $\mathcal{G}_{\mathrm{g}} \geq 0$. A smaller $\mathcal{G}_{\mathrm{g}}$ corresponds to better generalization performance.

Amortization gap. VAEs use amortized inference, i.e., they set the variational parameters of $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ for a given $\boldsymbol{x}$ to the output of the encoder $f_\phi(\boldsymbol{x})$. At test time, we can further maximize the ELBO over the individual variational parameters for each $\boldsymbol{x}$, which is more expensive but typically results in a better variational distribution $q^*(\boldsymbol{z} \mid \boldsymbol{x})$. We then study the *amortization gap* (Cremer et al., 2018),

$$\mathcal{G}_{\mathrm{a}} = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\mathrm{test}}}[\mathrm{ELBO}^*_\theta(\boldsymbol{x})\big] - \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\mathrm{test}}}[\mathrm{ELBO}_\Theta(\boldsymbol{x})\big] \geq 0 \tag{4}$$

where we replaced q by q^* in ELBO^*_θ. The encoder of a VAE tends to be more susceptible to overfitting than the decoder (Wu et al., 2017; Cremer et al., 2018; Shu et al., 2018). When the encoder overfits, its inference ability might not generalize to test data, which results in a lower ELBO and larger amortization gap $\mathcal{G}_{\mathrm{a}}$. A smaller $\mathcal{G}_{\mathrm{a}}$ corresponds to better generalization performance of the encoder.

Robustness gap. An overfitted encoder $f_\phi(\boldsymbol{x})$ potentially learns a less smooth function such that a small change in the input space can lead to a large difference in the output space. Hence, it is easier to construct an adversarial sample $\boldsymbol{x}^{\mathrm{a}} = \boldsymbol{x}^{\mathrm{r}} + \boldsymbol{\epsilon}$ ($\|\boldsymbol{\epsilon}\|$ is within a given attack radius δ) from a real data point $\boldsymbol{x}^{\mathrm{r}} \in \mathcal{D}_{\mathrm{test}}$. We construct adversarial samples by maximizing the symmetrized KL-divergence between $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{\mathrm{r}})$ and $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{\mathrm{a}})$ (Kuzina et al., 2022). For a successful attack, the reconstruction $\tilde{\boldsymbol{x}}^{\mathrm{a}} = g_\theta(\boldsymbol{z}^{\mathrm{a}}), \boldsymbol{z}^{\mathrm{a}} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{\mathrm{a}})$, is very different from the real data reconstruction $\tilde{\boldsymbol{x}}^{\mathrm{r}} = g_\theta(\boldsymbol{z}^{\mathrm{r}}), \boldsymbol{z}^{\mathrm{r}} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{\mathrm{r}})$, even though the inputs $\boldsymbol{x}^{\mathrm{a}}$ and $\boldsymbol{x}^{\mathrm{r}}$ are similar. Using the image similarity metric MS-SSIM (Wang et al., 2003), where higher MS-SSIM means more similar images, we define the *robustness gap* as

$$\mathcal{G}_{\mathrm{r}} = \mathbb{E}_{\boldsymbol{x}^{\mathrm{a}} \sim p(\boldsymbol{x}^{\mathrm{a}} \mid \boldsymbol{x}^{\mathrm{r}})}\, \mathbb{E}_{\boldsymbol{x}^{\mathrm{r}} \sim \mathcal{D}_{\mathrm{test}}}\big[\text{MS-SSIM}\left[\boldsymbol{x}^{\mathrm{r}}, \boldsymbol{x}^{\mathrm{a}}\right] - \text{MS-SSIM}\left[\tilde{\boldsymbol{x}}^{\mathrm{r}}, \tilde{\boldsymbol{x}}^{\mathrm{a}}\right]\big]. \tag{5}$$

A more robust VAE makes attacks difficult, i.e., it has a higher MS-SSIM$[\tilde{\boldsymbol{x}}^{\mathrm{r}}, \tilde{\boldsymbol{x}}^{\mathrm{a}}]$ and thus a smaller $\mathcal{G}_{\mathrm{r}}$ than a less robust VAE. For more details on the construction of the attack see Appendix A.

3 Related Work

We group related work into generalization and overparameterization, using diffusion models, and closing the performance gaps. Work on augmentation and distillation is discussed in Section 4.

Overparameterized models generalize. The classical theory of machine learning suggests not to over-increase model parameters (relative to the size of the training set) to avoid fitting spurious patterns or noise in the training data, which is known as overfitting and can lead to poor generalization. However, empirical evidence in deep learning suggests that overparameterized models generalize well (Bartlett et al., 2020). Specifically, when we keep increasing the number of parameters, after the first descent, the test error will increase but a second descent will follow. This phenomenon is known as *double descent* (Belkin et al., 2019). Understanding generalization in overparameterized models is still an active research direction (Brutzkus & Globerson, 2019; Allen-Zhu et al., 2019). Double descent has been demonstrated in many deep learning models (Nakkiran et al., 2021). Our work investigates both double descent and the effect of overparameterization in the setting of VAEs by analyzing two different sets of parameters.

Use samples from pre-trained diffusion models. There are many recent attempts to solve various tasks with data generated by diffusion models. Azizi et al. (2023) fine-tuned a text-to-image diffusion model on ImageNet, generated state-of-the-art samples with class labels, and trained a classifier on the samples. Their result shows that the classifier trained on generated data does not outperform the classifier trained on real data. In the adversarial training setting, using generated data by diffusion models shows significant improvements on classification robustness (Croce et al., 2021; Wang et al., 2023). Tian et al. (2023) found that the visual representations learned from samples generated by text-to-image diffusion models outperform the representations learned by SimCLR (Chen et al., 2020) and CLIP (Radford et al., 2021). Alemohammad et al. (2024) trained new diffusion models with samples from previously trained diffusion models, and they found that their sample quality and diversity progressively decrease. In this work, we find that using diffusion models as data sources improves the generalization performance of VAEs.

Improve generalization, amortized inference, and robustness in VAEs. Cremer et al. (2018) study the amortization gap in VAEs, and they notice that overfitting in the encoder is one of the contributing factors of the gap, and that it hurts generalization. Subsequent works try to close the amortization gap by introducing new inference techniques or procedures (Marino et al., 2018; Shu et al., 2018; Zhao et al., 2019). To close the generalization gap and reduce encoder overfitting, Zhang et al. (2022) propose to freeze the decoder after a certain amount of training steps, but further train the encoder by using reconstruction samples as part of the training data. As for adversarial robustness, Kuzina et al. (2022) propose to defend a pre-trained VAE by running MCMC during inference to move $\boldsymbol{z}$ towards "safer" regions in the latent space. In our case, both using synthetic data and overparameterization do not require changing the original inference procedure. They can be used orthogonally and take into account all three gaps simultaneously.

4 Two Paths Towards Generalization

Given a VAE that overfits and generalizes poorly, we want to know whether using more training data (**method A**); or more model parameters (**method B**) can improve its generalization performance and performance gaps (see Section 2). Here, we compare the two approaches, and discuss how using samples generated from a diffusion model is fundamentally different from naive data augmentation.

4.1 Method A: More Data - Diffusion Model as a $p_{\text{data}}(\boldsymbol{x})$

An ideal training objective for VAEs is to maximize

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\text{ELBO}_\Theta(\boldsymbol{x})\right]. \tag{6}$$

However, we only have $\mathcal{D}_{\text{train}}$ as a finite approximation of $p_{\text{data}}(\boldsymbol{x})$. Hence, we normally maximize

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}\sim \mathcal{D}_{\text{train}}}\left[\text{ELBO}_\Theta(\boldsymbol{x})\right], \tag{7}$$

Table 1: Training distributions for VAEs (see Figure 1), and whether we (1) can sample arbitrarily many unique samples from (2) an accurate approximation of $p_{\text{data}}(\boldsymbol{x})$.

approx. by	$\mathcal{D}_{\text{train}}$	$p_{\text{aug}}(\boldsymbol{x}')$	$p_{\text{DM}}(\boldsymbol{x}')$
(1) ∞ unique samples	✗	✓	✓
(2) accurate	✓	✗	✓

which can lead to overfitting. Rather than focusing on model architectures or training techniques as in prior works (Shu et al., 2018; Zhao et al., 2019; Zhang et al., 2022), we aim to mitigate overfitting by seeking a better approximation for $p_{\text{data}}(\boldsymbol{x})$ than $\mathcal{D}_{\text{train}}$. An ideal training data distribution should enable us to

(1) sample an unlimited amount of **unique samples** to avoid overfitting; and it should be

(2) **an accurate approximation of** $p_{\text{data}}(\boldsymbol{x})$, i.e., we are indeed modeling $p_{\text{data}}(\boldsymbol{x})$ rather than some different distribution (in practice, it needs to be an accurate model of $\mathcal{D}_{\text{train}}$).

Our hypothesis is that a good diffusion model[1] that has been pre-trained on $\mathcal{D}_{\text{train}}$ satisfies these two criteria: (1) we can generate unlimited samples from it, and (2) its training objective is designed to model $p_{\text{data}}(\boldsymbol{x})$. Therefore, we investigate training VAEs using a pre-trained diffusion model $p_{\text{DM}}(\boldsymbol{x}')$ instead of $\mathcal{D}_{\text{train}}$ as an approximation of the underlying distribution $p_{\text{data}}(\boldsymbol{x})$, i.e., by maximizing

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}' \sim p_{\text{DM}}(\boldsymbol{x}')} \left[\text{ELBO}_{\Theta}(\boldsymbol{x}') \right]. \tag{8}$$

We denote this method DMaaPx, short for "Diffusion Model as a $p_{\text{data}}(\boldsymbol{x})$". Figure 1 illustrates the intuition behind DMaaPx. The blue dots represent the finite data set $\mathcal{D}_{\text{train}}$. They are i.i.d. samples from the unknown underlying data distribution $p_{\text{data}}(\boldsymbol{x})$ which is shown by the dark-edged region. The green regions represent $p_{\text{DM}}(\boldsymbol{x}')$. We use shaded areas (green), not dots, to highlight that $p_{\text{DM}}(\boldsymbol{x}')$ models a continuous distribution which we can generate infinitely many samples from.

Diffusion models for data types other than images are less explored and might not accurately approximate $p_{\text{data}}(\boldsymbol{x})$ and therefore might not satisfy criterion (2). Moreover, due to the data processing inequality, information on $p_{\text{data}}(\boldsymbol{x})$ captured by a diffusion model trained on $\mathcal{D}_{\text{train}}$ cannot exceed the information contained in $\mathcal{D}_{\text{train}}$ (but for injected information via inductive biases). In reality, state-of-the-art diffusion models cannot fit $\mathcal{D}_{\text{train}}$ perfectly. Many recent works observe in both image and text settings that training generative models on generated data degrades the overall performance (Alemohammad et al., 2024; Bohacek & Farid, 2023; Bertrand et al., 2024; Shumailov et al., 2023). Hence, the continuity we gain by replacing $\mathcal{D}_{\text{train}}$ with $p_{\text{DM}}(\boldsymbol{x}')$ comes at the cost of potentially losing some amount of information contained in $\mathcal{D}_{\text{train}}$.

4.1.1 Difference Between Data Augmentation and DMaaPx

Data augmentation[2] has a similar goal as DMaaPx as both approaches aim to increase amount and diversity of training data. Training a VAE via data augmentation amounts to maximizing

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\text{train}}} \left[\mathbb{E}_{p_{\text{aug}}(\boldsymbol{x}' \mid \boldsymbol{x})} \left[\text{ELBO}_{\Theta}(\boldsymbol{x}') \right] \right], \tag{9}$$

where $p_{\text{aug}}(\boldsymbol{x}' \mid \boldsymbol{x})$ is a distribution over some hand-crafted transformations (e.g., scalings and rotations) of $\boldsymbol{x} \sim \mathcal{D}_{\text{train}}$.

Like DMaaPx, data augmentation replaces $\mathcal{D}_{\text{train}}$ with a continuous distribution $p_{\text{aug}}(\boldsymbol{x}') = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\text{train}}}[p_{\text{aug}}(\boldsymbol{x}' \mid \boldsymbol{x})]$. But $p_{\text{aug}}(\boldsymbol{x}')$ can be a less accurate approximation of $p_{\text{data}}(\boldsymbol{x})$ than $p_{\text{DM}}(\boldsymbol{x})$ (Table 1). Firstly, typical data augmentation techniques generate new training points $\boldsymbol{x}'$ by conditioning on a *single* original data point $\boldsymbol{x}$ (i.e., the nested expectations in Eq. (9)). Thus, $p_{\text{aug}}(\boldsymbol{x}')$ cannot interpolate between points in $\mathcal{D}_{\text{train}}$ (i.e., the gaps between purple regions in Figure 1). By contrast, in DMaaPx, each training data point $\boldsymbol{x}' \sim p_{\text{DM}}(\boldsymbol{x}')$ is drawn from a diffusion model trained on the entire dataset $\mathcal{D}_{\text{train}}$. Hence, each $\boldsymbol{x}'$ is effectively conditioned on the full $\mathcal{D}_{\text{train}}$.

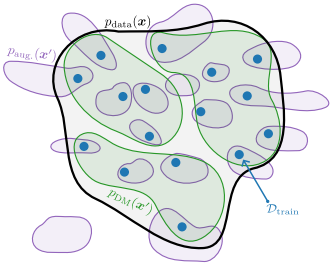


Figure 1: Training distributions. $p_{\text{aug}}(\boldsymbol{x}') = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_{\text{train}}}[p_{\text{aug}}(\boldsymbol{x}' \mid \boldsymbol{x})]$ only extrapolates from individual data points $\boldsymbol{x} \sim \mathcal{D}_{\text{train}}$ and has density outside the support of $p_{\text{data}}(\boldsymbol{x})$ (e.g., when flipping the digit "2"). By contrast, $p_{\text{DM}}(\boldsymbol{x}')$ can interpolate between data $\boldsymbol{x} \sim \mathcal{D}_{\text{train}}$.

Secondly, the transformations used for $p_{\text{aug}}(\boldsymbol{x}' \mid \boldsymbol{x})$ are drawn from a manually curated catalog. This catalog is heavily based on prior assumptions regarding *invariances* in the data type under consideration, which can introduce bias. In

[1] An *unconditional* diffusion model, as opposed to a conditional one like Stable Diffusion (Rombach et al., 2022).

[2] Using samples from generative models for training is sometimes considered as a kind of data augmentation in the context of supervised learning (Yang et al., 2022). In the present work, we deliberately separate generative models from the broader sense of data augmentation, and we consider data augmentation in a narrow sense such that the augmented data is an output of some transformation conditioned on an single $\boldsymbol{x} \in \mathcal{D}_{\text{train}}$.

practice, one has to make assumptions and decide whether the (unknown) true data distribution $p_{\text{data}}(\boldsymbol{x})$ is invariant under the considered transformations. For instance, with images, we assume invariance to minor translations, hue shifts, and zooms. For data types other than images, such as neural activations or molecules, manually crafting the appropriate set of augmentations might be less obvious. This manual design of augmentations may result in problems of (i) not modeling the *full* extend of $p_{\text{data}}(\boldsymbol{x})$ or (ii) modeling density *outside* of $p_{\text{data}}(\boldsymbol{x})$. Figure 1 depicts both: problem (i) corresponds to "empty" space between areas of $p_{\text{aug}}(\boldsymbol{x}')$ (purple); problem (ii) corresponds to density of $p_{\text{aug}}(\boldsymbol{x}')$ outside of $p_{\text{data}}(\boldsymbol{x})$. DMaaPx eliminates these assumptions, which makes the method more resilient against human bias (but less interpretable).

In summary, while traditional data augmentation introduces diversity based on *invariances* in the data, DMaaPx uses an expressive generative model to extrapolate from the *empirical* diversity of the data.

4.1.2 A Distillation Perspective

Using samples from a pre-trained diffusion model to train a VAE can also be viewed from a distillation perspective (Hinton et al., 2015). Distillation describes the process of transferring knowledge from a large model to a smaller one. In practice, distillation is often used because a smaller model is *less expensive* to be deployed in production. On the contrary, here we consider transferring knowledge between models designed with *different modeling assumptions or structures*. We refer to this as *cross-model-class distillation*, and the conventional usage of distillation as *within-model-class distillation*. While both types seek to transfer knowledge from a source to a target model, cross-model-class distillation emphasizes more on enhancing functionalities that are unique to the target model rather than mirroring the capabilities shared with the source model. For instance, Frosst & Hinton (2017) distill a neural network into a soft decision tree to improve its unique capability of providing explanations to decisions.

DMaaPx belongs to cross-model-class distillation, i.e., it distills diffusion models to VAEs. The goal of DMaaPx is not to rival diffusion models in sample quality, but rather to improve the unique capabilities of VAEs such as its variational bottleneck (Alemi et al., 2016) for learning fast, controllable, and interpretable representations (see Section 1 for a discussion). Hence, DMaaPx fundamentally differs from approaches that train VAEs on samples produced by VAEs (Shumailov et al., 2023), or diffusion models on outputs of diffusion models (Alemohammad et al., 2024), which belongs to within-model-class distillation.

4.2 Method B: More Parameters

The success of deep learning and empirical evidence (Nakkiran et al., 2021) show that increasing the number of parameters in a model can improve its generalization performance. However, increasing the number of parameters in one part of the model can have a different effect than increasing the number of parameters in another part of the model. Different sets of parameters form distinct *complexity axes*, along which generalization performance can behave in different ways.

In practice, increasing the number of parameters in VAEs is usually done by changing two hyperparameters, which control two distinct sets of model parameters, $\Theta = \Theta_{\boldsymbol{z}} \cup \Theta_{\neg \boldsymbol{z}}$. Here, $\Theta_{\boldsymbol{z}}$ contains the parameters that determine the latent dimension $\mathrm{d}_{\boldsymbol{z}}$ and interact directly with $\boldsymbol{z}$, i.e., the weights in the last layer of the encoder $f_\phi(\boldsymbol{x})$ and the first layer of the decoder $g_\theta(\boldsymbol{z})$. Changing the number $|\Theta_{\boldsymbol{z}}|$ of parameters in $\Theta_{\boldsymbol{z}}$ changes the dimension $\mathrm{d}_{\boldsymbol{z}}$ of $\boldsymbol{z}$. We denote the remaining parameters as $\Theta_{\neg \boldsymbol{z}}$.

We investigate the effect of increasing $|\Theta_{\boldsymbol{z}}|$ and $|\Theta_{\neg \boldsymbol{z}}|$ separately as the increase has different implications for each parameter group. For both sets, we only consider changing the width of the VAE, not the depth. For a VAE with mean field assumption, for example, increasing $|\Theta_{\boldsymbol{z}}|$ implies that we update our belief about the underlying data generative process, which we now believe to involve more independent factors (hence, we increase $\mathrm{d}_{\boldsymbol{z}}$ to model these additional independent factors).

Explaining Double Descent with Multiple Complexity Axes. With the two complexity axes $|\Theta_{\boldsymbol{z}}|$ and $|\Theta_{\neg \boldsymbol{z}}|$ mentioned above, we might potentially observe the phenomenon of double descent in VAEs. Recent work by Curth et al. (2023) tries to explain double descent (see Section 1) in non-deep models such as trees and linear regression. They suggest that double descent is the result of plotting the test error along

an aggregated model complexity axis (i.e., the total number of parameters), even though the raw parameters are increased first along one complexity axis and then along another. In other words, the test error along each axis shows the classical U-shape, but plotting them into the same model complexity axis leads to the shape of double descent. We refer readers to (Curth et al., 2023, Figure 1) for an illustration. Whether this explanation applies to *deep* double descent is still unknown. Nevertheless, it highlights that increasing different sets of model parameters can impact the model differently.

5 Experiments and Discussions

In this section, we introduce the experimental setup and evaluate the generalization performance as well as the performance gaps (see Section 2) for both using more training data (**method A**) and more parameters (**method B**). We then evaluate and discuss the effects of these two methods.

5.1 Experimental Setup

Datasets. We evaluate both methods A & B, on CIFAR-10 (Krizhevsky et al., 2009). We further evaluate method A on BinaryMNIST (LeCun et al., 1998) and FashionMNIST (Xiao et al., 2017), which show consistent results with CIFAR-10 (see Appendix B.3). Due to computation constraints, we did not evaluate method B on these two easier datasets. As a preparation for DMaaPx in method A, we train a diffusion model $p_{\mathrm{DM}}(\boldsymbol{x}')$ on each training set $\mathcal{D}_{\mathrm{train}}$, which we then use to generate the training data for the VAEs. We use the implementation by Karras et al. (2022). Further details can be found in Appendix D.

VAE architectures. We assume fixed Gaussian priors $p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. For the conditional likelihood $p_\theta(\boldsymbol{x} \,|\, \boldsymbol{z})$, we use a discretized mixture of logistics (MoL; (Salimans et al., 2017)). For the inference model $q_\phi(\boldsymbol{z} \,|\, \boldsymbol{x})$, we use fully factorized Gaussian distributions whose means and variances are the outputs of the encoder. Both the encoder and the decoder consist of convolutional layers with residual connections. The number of output channels for the encoder and the number of input channels for the decoder (i.e., the latent dimension of $\boldsymbol{z}$) is denoted as $\mathrm{d}_{\boldsymbol{z}} = m_z \times 64$, where m_z is an integer multiplier. The number of the internal channels, which governs the rest of the parameters, is denoted as n_c. Hence, $|\Theta_{\boldsymbol{z}}| = \mathrm{d}_{\boldsymbol{z}} \times c_1$ and $|\Theta_{\neg \boldsymbol{z}}| = n_c \times c_2$, where c_1 and c_2 are the appropriate constants. For more details on the network architectures, hyperparameters, and training please consult Appendix B.1.

Baselines. The default models for method A and the baseline for method B use $m_z = 1$ and $n_c = 256$. For method A, we compare VAEs trained with DMaaPx against three other models trained on: (i) repetitions of $\mathcal{D}_{\mathrm{train}}$ ("Normal Training"); (ii) carefully tuned augmentation for $\mathcal{D}_{\mathrm{train}}$ ("Aug.Tuned"); and (iii) plausible augmentation for images in general ("Aug.Naive"). The results are averaged over 3 random seeds. Note that "Aug.Naive" is not tuned towards a given training dataset $\mathcal{D}_{\mathrm{train}}$ and can result in out-of-distribution data, e.g. a horizontally flipped digit "2" for MNIST. This mimics situations where the invariances of the data modality are less clear than for images. We report the specific augmentations used for Aug.Tuned and Aug.Naive in Appendix E. For method B, we compare VAEs trained with various $m_z \in \{1, \dots, 64\}$ and $n_c \in \{1, \dots, 512\}$. When documenting the training progress, the term "epoch" usually refers to one complete pass over $\mathcal{D}_{\mathrm{train}}$. For DMaaPx, this term is not applicable since it can sample unlimited data from $p_{\mathrm{DM}}(\boldsymbol{x}')$. Therefore, we measure training progress of DMaaPx in "effective epochs", which count multiples of $|\mathcal{D}_{\mathrm{train}}|$ training samples. We train all models for 1000 (effective) epochs.

5.2 Training with synthetic data improves generalization and minimizes the gaps

Figure 2 shows the performance for generalization (left), amortized inference (mid), and robustness (right) across normal training, augmentations, and DMaaPx. We observe that DMaaPx has the highest ELBO on $\mathcal{D}_{\mathrm{test}}$ (left, solid green) and the smallest generalization, amortization, and robustness gaps (Eqs. 3-5). Augmentation provides competitive improvements, but is not as good as DMaaPx. This implies that VAEs trained with DMaaPx approximate the underlying distribution $p_{\mathrm{data}}(\boldsymbol{x})$ better than those trained on $\mathcal{D}_{\mathrm{train}}$ solely, or on augmented data, as suspected in Section 4.1.1.

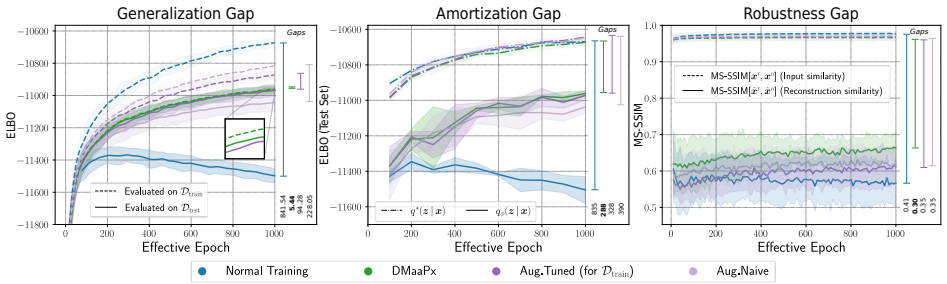


Figure 2: Generalization (left), amortized inference (mid), and robustness (right) performance for VAEs trained with Eqs. (7)-(9). Being slightly better than augmentations, DMaaPx consistently has the best performance on the test set and the smallest generalization, amortization, and robustness gap.

We also evaluate whether the improvements in generalization from DMaaPx and augmentation propagate to downstream tasks for which VAEs are better suited than diffusion models. We assess the representation learning performance by classifying the mean $\boldsymbol{\mu}$ of $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ for each $\boldsymbol{x}$. We also test both the quality and robustness of the learned representations by using CIFAR-10-C (Hendrycks & Dietterich, 2019), a dataset containing 19 versions of the CIFAR-10 test set, each corrupted with a different corruption. For every test set, we encode each image to its $\boldsymbol{\mu}$ and then split all representations into two separate subsets. We use one subset to train a classifier and test on the other. Our experiments include four classifiers: logistic regression (LR), a support vector machine (Boser et al., 1992) with linear kernel (SVM-L), an SVM with radial basis function kernel (SVM-RBF), and k-nearest neighbors (kNN) with $k = 5$. Figure 4 (left) shows that classifiers trained with representations from DMaaPx (green) outperform normal training on average (details in Appendix G). By contrast, augmentation (purple) hurts performance.

Comparison to Prior Work on Modifying the Training Procedures. In Figure 5 we compare the generalization performance of DMaaPx to the previously proposed Reverse Half Asleep (RHA) method of (Zhang et al., 2022). Unlike DMaaPx, which focuses on the training data, RHA modifies the training procedure of a VAE such that first a VAE is trained 500 epochs, and then the decoder is fixed on only the encoder is trained on samples of the decoder for an additional 500 epochs. We find that DMaaPx, being conceptually much simpler, outperforms RHA significantly.

5.3 Smoothness of Latent Space

We hypothesize that an overfitted encoder often learns a less smooth $f_\phi(\boldsymbol{x})$ meaning that a small change in $\boldsymbol{x}$ can result in a large difference in the latent space. Here, we report an exemplary density for an interpolation in the latent space of a VAE trained on CIFAR-10 with normal training and DMaaPx. We map two data samples $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ from the test set of CIFAR-10 to their latent means using the encoder $\boldsymbol{z}_0 = \boldsymbol{\mu}_0 = f_\phi^{\boldsymbol{\mu}}(\boldsymbol{x}_0)$ where $f_\phi^{\boldsymbol{\mu}}(\boldsymbol{x})$ denotes the encoder returning only the mean of $q_\phi(\boldsymbol{x})$. We interpolate linearly in the latent space and evaluate $\log q_\phi(\boldsymbol{z}_\alpha \mid \tilde{\boldsymbol{x}}_\alpha)$ where $\boldsymbol{z}_\alpha = \alpha \boldsymbol{z}_0 + (1-\alpha)\boldsymbol{z}_1, \alpha \in [0,1]$ and $\tilde{\boldsymbol{x}}_\alpha$ is a reconstruction of $\boldsymbol{z}_\alpha$.

We find that, firstly, the density is higher for DMaaPx compare to normal training and, secondly, the latent space appears to be smoother for DMaaPx compared to normal training.

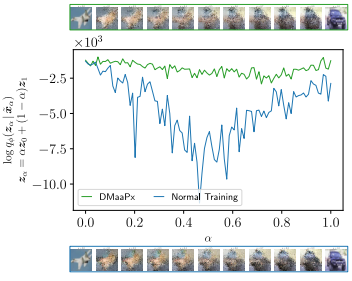


Figure 3: Density of $\log q_\phi$ evaluated on a line that linearly interpolates between two data samples from the test set of CIFAR-10. DMaaPx is smoother than normal training.

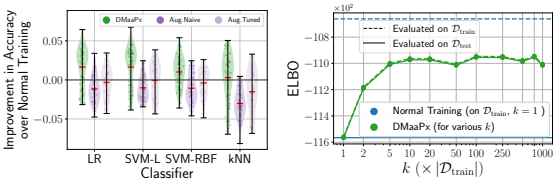


Figure 4: Left: Improvements in classification accuracy over normal training for various classifiers trained on the latent representations of CIFAR-10-C. Each column contains 19×3 points (i.e., 19 corruptions, each VAE trained with 3 random seeds). Right: Generalization performance as a function of the amount k of training data sampled from a diffusion model. Horizontal blue lines show baseline performance (VAE trained directly on $\mathcal{D}_{\text{train}}$). All VAEs were trained for 1000 effective epochs. $k \approx 10$ suffices.

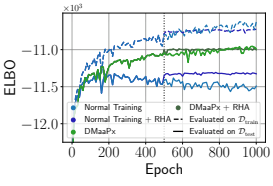


Figure 5: Comparison between DMaaPx and Reverse Half Asleep (RHA) (Zhang et al., 2022). Dotted vertical line shows epoch when decoder is frozen for RHA. "Normal Training + RHA" improves upon baseline but it is still worse than DMaaPx. Also, adding RHA on top of DMaaPx does not lead to improvements.

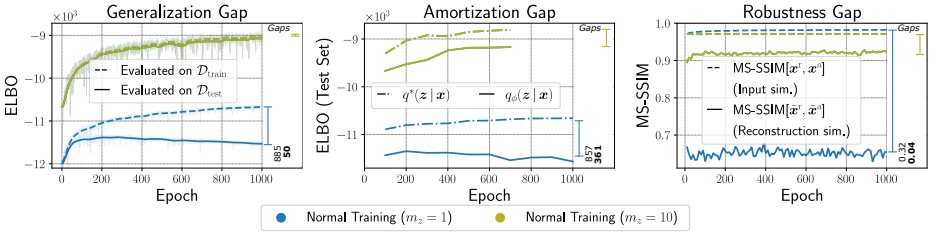


Figure 6: Generalization (left), amortized inference (mid), and robustness (right) performance for models with different m_z but the same $n_c = 256$, using normal training. Due to computation constraints, we only plot the amortization gap for $m_z = 10$ up to epoch 700.

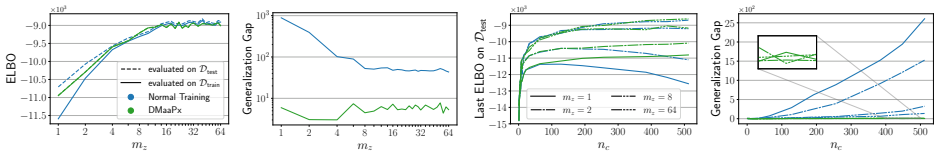
5.4 Do we need "unlimited" synthetic data?

With the diffusion model in DMaaPx, we can train a VAE on unlimited samples from $p_{\text{DM}}(\boldsymbol{x}')$, which improves various performance metrics as demonstrated above. However, generating lots of samples from a diffusion model is computationally expensive (see also Appendix D.2). Therefore, we further explore: "*Do we really need an infinite number of samples?*" The answer, reassuringly, is "*No*".

Figure 4 (right) shows the generalization performance of DMaaPx on CIFAR-10 where the amount of training data is restricted to $k \times |\mathcal{D}_{\text{train}}|$ with $k = 1, \ldots, 1000$. After k effective training epochs samples repeat. All models are trained for 1000 effective epochs. The horizontal blue lines represent the generalization gap of normal VAE training (on $\mathcal{D}_{\text{train}}$ and $k = 1$) at epoch 1000 from Figure 2 (left). For $k = 1$, DMaaPx matches normal training on CIFAR-10. The ELBO plateaus for $k \geq 10$, indicating that approximately $10 \times |\mathcal{D}_{\text{train}}|$ samples offer a similar generalization as $1000 \times |\mathcal{D}_{\text{train}}|$ samples.

5.5 Increasing $|\Theta_{\boldsymbol{z}}|$ improves generalization and minimizes the gaps

Figure 6 shows the generalization (left), amortized inference (mid), and robustness (right) performance between the baseline parameter setting ($m_z = 1$, $n_c = 256$) and the setting with an increased amount of parameters ($m_z = 10$, $n_c = 256$). Note that we only increase $|\Theta_{\boldsymbol{z}}|$ here and keep $|\Theta_{\neg \boldsymbol{z}}|$ unchanged. We observe that $m_z = 10$ has the higher ELBO on $\mathcal{D}_{\text{test}}$ (left), and the smaller generalization, amortization, and



(a) (b) (c) (d)

Figure 7: (a) and (b) show the changes of ELBOs and generalization gaps while fixing $n_c = 256$ and increasing m_z. (c) and (d) show the effects of increasing n_c while keeping m_z fixed. Besides normal training, we also evaluate DMaaPx in this setting.

robustness gaps (Eqs. 3-5). The resulting performance improvements are similar to DMaaPx in Figure 2, where we also see a higher ELBO test and smaller gaps.

We also investigate whether continuing to increase m_z will lead to better or worse generalization performance. Figure 7 (a, b) show trends of the ELBOs on $\mathcal{D}_{\text{test}}$ and the generalization gaps (Eq. (3)) as the result of increasing m_z from 1 to 64. We see that increasing m_z leads to higher test ELBO for normal training until $m_z = 16$, after which the test ELBO stays around the same level. We also find that the benefit of using synthetic data (i.e., DMaaPx) for improving test ELBO diminishes as m_z increases. In particular, after $m_z = 16$, both normal training and DMaaPx have similar test ELBOs. However, we want to highlight that DMaaPx has always a smaller generalization gap than normal training, even after $m_z = 16$. Small generalization gaps are desirable as they imply that the ELBO evaluated on the training set can be used to predict the performance of a VAE on a held-out test set.

5.6 Increasing $|\Theta_{\neg \boldsymbol{z}}|$ might hurt generalization

We further investigate the effect of increasing parameters along the other complexity axis (i.e., $|\Theta_{\neg \boldsymbol{z}}|$). Figure 7 (c, d) show the test ELBOs and the generalization gaps (Eq. (3)) as functions of $n_c = 1$ to 512. Subplot (c) shows that when $m_z = 1$ and $m_z = 2$, increasing n_c results in the classical U-shape of overfitting for normal training, where the test performance first grows then drops. Increasing m_z further to 8 and 64 tilts the right-hand side of the U-shaped curve upward and eliminates the symptom of declining generalization w.r.t. larger n_c, which aligns with Figure 7 (a). In particular, when $m_z = 64$, we see improved on generalization as we keep increasing n_c, even though the improvement from enlarging m_z is already saturated (as showed in Figure 7 (a)). This implies that: (1) the two sets of parameters $\Theta_{\boldsymbol{z}}$ and $\Theta_{\neg \boldsymbol{z}}$ do indeed have different meanings; (2) increasing $|\Theta_{\neg \boldsymbol{z}}|$ when $|\Theta_{\boldsymbol{z}}|$ is small hurts generalization performance; (3) increasing both $|\Theta_{\neg \boldsymbol{z}}|$ and $|\Theta_{\boldsymbol{z}}|$ leads to better generalization than increasing only one of them.

The plot for generalization gaps (Figure 7 (d)) shows that, by increasing n_c, the gaps for normal training become larger. However, increasing m_z reduces the gaps (which aligns with Figure 7 (b)). In addition, both plots (Figure 7 (c) and (d)) show that using synthetic data (i.e., DMaaPx) on top of adding parameters does not hurt and often improves generalization. Especially, using DMaaPx constantly results in near-zero generalization gaps (see the overlapping flat lines in Figure 7 (d)).

5.7 Double descent in VAEs?

Section 5.5 and Section 5.6 indicate that the modern wisdom of "larger models generalize better" does indeed apply to VAEs. Therefore, we are curious whether the double descent phenomenon also applies here. Figure 7 (a) and (c) show that if we fix one complexity axis and only increase parameters along the other complexity axis, the test ELBO exhibits either a U-shaped or a L-shaped curve, which does not look like double descent. However, one can artificially construct a double descent behavior if one sequentially increases parameters first along one axis and then along the other axis, and plots the test ELBO against the overall model complexity (i.e., the total number of parameters). This is illustrated in Figure 8. The figure shows the negative test ELBO, which shows three phases:

(1) we start with a fixed $m_z = 1$ and increase n_z from 1 to 256 (the cyan U-curve); (2) we then fixed $n_z = 256$ and start increasing m_z from 1 to 64 (the brown vertical line); (3) lastly, we fixed $m_z = 64$ and increase n_z from 256 to 512 (the cyan horizontal line). Phase (2) seems vertical due to the fact that increasing m_z by 1 adds much less raw parameters than increasing n_z by 1.

Now we can see indications of a double descent behavior as the negative test ELBO goes down, up, and down again as we keep increasing the total number of parameters in the model. While the double-descent behavior was enforced artificially here by the choice of trajectory in model complexity space, it is conceivable that a trained models may allocate resources in a similar way while following a more natural trajectory. This aligns with our discussion in Section 4.2 and the work by Curth et al. (2023), who point out that double descent in non-deep models is due to increasing parameters along distinct complexity axis sequentially but plotting them on a combined complexity axis.

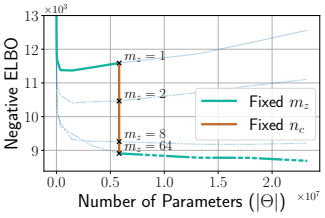


Figure 8: By increasing the total number of parameters first along the complexity axis $|\Theta_{\neg \mathbf{z}}|$ (i.e., larger n_c, cyan), then along the complexity axis $|\Theta_{\mathbf{z}}|$ (i.e., larger m_z, brown line), then back to $|\Theta_{\neg \mathbf{z}}|$ (cyan), we see indications of double descent.

6 Conclusion

In this paper, we study overfitting and generalization in VAEs by investigating the effects of (a) training with synthetic data and (b) increasing the number of model parameters, both of which are timely questions to be investigated in generative models. We find that training on samples from a diffusion model that was pre-trained on the training dataset consistently improves performance. Generally, increasing the amount of parameters also helps. However, when constrained by resources, one should prioritize the parameters related to the latent dimension over the parameters that do not directly impact the latent dimension. Applying (a) and (b) together can further improve the performance. Additionally, we find indications of a double-descent phenomenon in VAEs which we defer to future work for an investigation in detail.

Acknowledgments

The authors would like to thank Anna Kuzina, Zhen Liu, and Weiyang Liu for helpful discussions. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A. Robert Bamler acknowledges funding by the German Research Foundation (DFG) for project 448588364 of the Emmy Noether Programme. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Tim Z. Xiao and Johannes Zenn.

References

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations*, 2016. 2, 6

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard Baraniuk. Self-consuming generative models go MAD. In *International Conference on Learning Representations*, 2024. 1, 4, 5, 6

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 32, 2019. 4

Shekoofeh Azizi, Simon Kornblith, Chitwan Saharia, Mohammad Norouzi, and David J Fleet. Synthetic data from diffusion models improves imagenet classification. *arXiv preprint arXiv:2304.08466*, 2023. 4, 25

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations*, 2017. 29

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018. 1, 2

Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020. 4

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. 2, 4

Quentin Bertrand, Joey Bose, Alexandre Duplessis, Marco Jiralerspong, and Gauthier Gidel. On the stability of iterative retraining of generative models on their own data. In *International Conference on Learning Representations*, 2024. 1, 5

Sebastian Bischoff, Alana Darcher, Michael Deistler, Richard Gao, Franziska Gerken, Manuel Gloeckler, Lisa Haxel, Jaivardhan Kapoor, Janne K Lappalainen, Jakob H Macke, et al. A practical guide to sample-based statistical distances for evaluating generative models in science. *Transactions on Machine Learning Research*, 2024. 26

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 2017. 3

Matyas Bohacek and Hany Farid. Nepotistically trained generative-ai models collapse. *arXiv preprint arXiv:2311.12202*, 2023. 1, 5

Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, 1992. 8, 29

Alon Brutzkus and Amir Globerson. Why do larger models generalize better? a theoretical perspective via the xor problem. In *International Conference on Machine Learning*, pp. 822–830. PMLR, 2019. 4

Olmo Cerri, Thong Q Nguyen, Maurizio Pierini, Maria Spiropulu, and Jean-Roch Vlimant. Variational autoencoders for new physics mining at the large hadron collider. *Journal of High Energy Physics*, 2019 (5):1–29, 2019. 1, 2

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020. 4

Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7939–7948, 2020. 1, 2

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*, 2018. 1, 3, 4

Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In *Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. 4, 25

Alicia Curth, Alan Jeffares, and Mihaela van der Schaar. A u-turn on double descent: Rethinking parameter counting in statistical learning. *arXiv preprint arXiv:2310.18988*, 2023. 2, 6, 7, 11

Daniel Flam-Shepherd, Tony C Wu, Xuemei Gu, Alba Cervera-Lierta, Mario Krenn, and Alan Aspuru-Guzik. Learning interpretable representations of entanglement in quantum optics experiments using deep generative models. *Nature Machine Intelligence*, 4(6):544–554, 2022. 1, 2

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017. 6

Rabia Gondur, Usama Bin Sikandar, Evan Schaffer, Mikio Christian Aoi, and Stephen L Keeley. Multi-modal gaussian process variational autoencoders for neural and behavioral data. *arXiv preprint arXiv:2310.03111*, 2023. 1, 2

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 17, 26

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. 8, 30

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, 2017. 29

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 6

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020. 2

Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Advances in Neural Information Processing Systems*, 2020. 27

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, 2022. 7, 25

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014. 1

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 7, 25

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22, 1951. 17

Anna Kuzina, Max Welling, and Jakub Tomczak. Alleviating adversarial attacks on variational autoencoders with mcmc. In *Advances in Neural Information Processing Systems*, 2022. 2, 3, 4, 17

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998. 7, 25

Joe Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. In *International Conference on Machine Learning*, 2018. 4

David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems*, 2018. 1, 2

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021. 2, 4, 6

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2018. 19

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. 4

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014. 1

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 1951. 26

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 5

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323, 1986. 26

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. PixelCNN++: Improving the pixelCNN with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2017. 7, 17

Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport. In *International Conference on Learning Representations*, 2018. 29

Rui Shu, Hung H Bui, Shengjia Zhao, Mykel J Kochenderfer, and Stefano Ermon. Amortized inference regularization. In *Advances in Neural Information Processing Systems*, 2018. 1, 3, 4

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. The curse of recursion: Training on generated data makes models forget. *arXiv preprint arxiv:2305.17493*, 2023. 1, 5, 6

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015. 2

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. 2, 25

Lucas Theis, van den Oord Aäron, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016. 29

Yonglong Tian, Lijie Fan, Phillip Isola, Huiwen Chang, and Dilip Krishnan. Stablerep: Synthetic images from text-to-image models make strong visual representation learners. *arXiv preprint arXiv:2306.00984*, 2023. 4, 25

Ba-Hien Tran, Giulio Franzese, Pietro Michiardi, and Maurizio Filippone. One-line-of-code data mollification improves optimization of likelihood-based generative models. In *Advances in Neural Information Processing Systems*, 2023. 21

Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. Better diffusion models further improve adversarial training. In *International Conference on Machine Learning*, 2023. 4, 25

Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, 2003. 3

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. In *International Conference on Learning Representations*, 2017. 1, 3

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 7, 25

Tim Z. Xiao and Robert Bamler. Trading information between latents in hierarchical variational autoencoders. In *International Conference on Learning Representations*, 2023. 29

Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Furao Shen. Image data augmentation for deep learning: A survey. *arXiv preprint arXiv:2204.08610*, 2022. 5

Yibo Yang, Stephan Mandt, Lucas Theis, et al. An introduction to neural data compression. *Foundations and Trends® in Computer Graphics and Vision*, 2023. 1, 2

Mingtian Zhang, Peter Hayes, and David Barber. Generalization gap in amortized inference. In *Advances in Neural Information Processing Systems*, 2022. 4, 8, 9

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Balancing learning and inference in variational autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019. 4

Ding Zhou and Xue-Xin Wei. Learning identifiable and interpretable latent models of high-dimensional neural activity using pi-vae. In *Advances in Neural Information Processing Systems*, 2020. 1, 2

Appendix

Table of Contents

A Details on The Adversarial Attack

We follow Kuzina et al. (Kuzina et al., 2022) and construct an unsupervised encoder attack that optimizes the pertubation $\boldsymbol{\epsilon}$ to incur the largest possible change in $q_\phi(\cdot \mid \boldsymbol{x})$,

$$\boldsymbol{\epsilon} = \underset{\|\boldsymbol{\epsilon}\|_\infty \leq \delta}{\arg\max}\ \mathrm{SKL}\left[q_\phi(\cdot \mid \boldsymbol{x}^{\mathrm{r}} + \boldsymbol{\epsilon}) \parallel q_\phi(\cdot \mid \boldsymbol{x}^{\mathrm{r}})\right] \tag{10}$$

where SKL denotes the symmetric Kullback-Leibler divergence (Kullback & Leibler, 1951). We optimize $\boldsymbol{\epsilon}$ for n^ϵ iterations with projected gradient descent utilizing a learning rate of η. The robustness gap (see Section 2) is computed over n^{r} real images and n^{a} random seeds. The exact hyperparameters can be found in Table 2.

Table 2: Hyperparameters for the unsupervised encoder attack.

	BinaryMNIST	FashionMNIST	CIFAR-10
n^{r}	50	50	20
n^{a}	10	10	10
n^ϵ	50	50	100
η	1.0	1.0	1.0
δ	0.1	0.1	0.05

B Further Details on VAEs and Ablations

In this section we present further details on the VAE models and ablation studies.

B.1 Architectures and Training Cost

This section provides a detailed description of the VAE models utilized throughout the paper.

Table 3: Details on VAE architectures ordered by dataset. MoL refers to the discretized mixture of logistics likelihood model (Salimans et al., 2017).

dataset	likelihood	architecture
BinaryMNIST	Bernoulli	fully-connected
FashionMNIST	fixed-variance Gaussian	fully-connected
FashionMNIST	MoL	fully-connected
CIFAR-10	MoL	residual network

We consider a fully-connected architecture and a residual architecture (He et al., 2016). Table 3 gives more details on the likelihood model and architecture. For all VAEs, we choose the hyperparameters of the VAE models by consulting existing literature. The MoL likelihood for FashionMNIST is discussed in Appendix B.

The fully-connected architecture maps from an input dimension of 32^2 to a hidden dimension of 512. After a hidden layer mapping from 512 to 512, the output is mapped to a latent variable of dimension 16. The decoder mirrors the encoder and maps the latent variable of dimension 16, via three layers, to the original input size.

For the residual architecture we present specific tables for the encoder (Table 4), decoder (Table 5), and the residual block (Table 6).

We train each VAE on a single NVIDIA GeForce RTX 2080 Ti 11GB GPU.

Table 4: Residual encoder architecture.

layer	dimension	additional parameters
Convolution	$3 \to n_c$	kernel size: 4, stride: 2, padding: 1, no bias
BatchNorm	n_c	
ReLU		
Convolution	$n_c \to n_c$	kernel size: 4, stride: 2, padding: 1, no bias
BatchNorm	n_c	
Residual	n_c	
Residual	n_c	
Convolution	$n_c \to 2d_z$	kernel size: 1, stride: 1, padding: 0, bias

Table 5: Residual decoder architecture.

layer	dimension	additional parameters
Convolution	$d_z \to n_c$	kernel size: 1, stride: 1, padding: 0, no bias
BatchNorm	n_c	
Residual	n_c	
Residual	n_c	
Transposed Convolution	$n_c \to n_c$	kernel size: 4, stride: 2, padding: 1, no bias
BatchNorm	n_c	
ReLU		
Convolution	$n_c \to 3$	kernel size: 1, stride: 1, padding: 0, bias

Table 6: Architecture of a residual block.

layer	dimension	additional parameters
ReLU		
Convolution	$n_c \to n_c$	kernel size: 3, stride: 1, padding: 1, no bias
BatchNorm	n_c	
ReLU		
Convolution	$n_c \to n_c$	kernel size: 3, stride: 1, padding: 1, no bias
BatchNorm	n_c	

B.2 Reconstructions

Figure 9 shows reconstructions for CIFAR-10. We reconstruct a random subset of 9 samples from the test set. The VAEs we use are quite small and simple, they are also non-hierachical, which makes it hard to model CIFAR-10 well. Therefore, qualitatively the differences between the methods are quite subtle. But still, we can see that the DMaaPx's reconstruction is slightly better for the rightmost image in the second row.

B.3 Additional Datasets

Figure 10 shows the three generalization gaps for the BinaryMNIST dataset. Figure 11 shows the three generalization gaps for the Fashion-MNIST dataset. All claims that have been made in the main text also hold for these datasets.

Figure 9: Reconstructions for CIFAR-10 for a random subset of 9 samples from the test set.

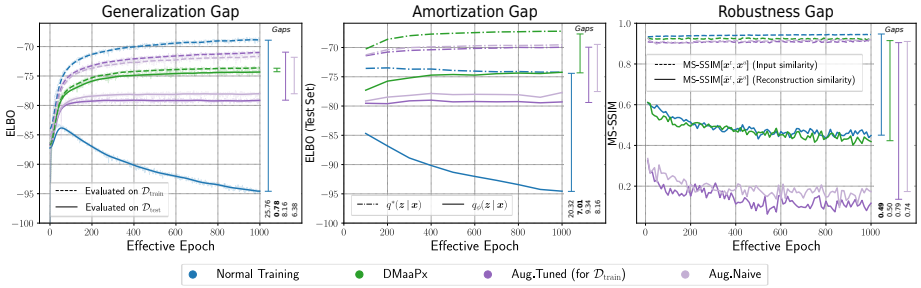


Figure 10: Generalization gap, amortization gap, and robustness gap on the BinaryMNIST dataset.

B.4 Different Conditional Likelihoods

VAEs' modeling assumptions for the conditional likelihood $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ often differ based on data or use case. While a Gaussian likelihood is used for applications that focus on low reconstruction error (e.g., lossy data compression), a MoL likelihood is used if the density of the data matters (e.g., generative modeling or lossless data compression). While the experiments in the main text considered a MoL likelihood, the model trained on FashionMNIST (Figure 11) uses a Gaussian likelihood and the model trained on BinaryMNIST (Figure 10) uses a Bernoulli likelihood (also compare Appendix B.1). Figure 12 also evaluates MoL for FashionMNIST, and we observe a similar behavior as in its Gaussian counterpart in Figure 11 (left). In summary, DMaaPx is less prone to overfitting than normal training and augmentation, for all investigated likelihoods.

B.5 Training ELBO versus ELBO on $\mathcal{D}_{\text{train}}$

Remark (Test data entropy can also affect the ELBO value). Note that from Eqs. (2) and (1), we have

$$\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\text{ELBO}_\Theta(\boldsymbol{x})\right] \;\leq\; \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] \;=\; -H[p(\boldsymbol{x}), p_\theta(\boldsymbol{x})] \leq -H[p(\boldsymbol{x})], \tag{11}$$

where H denotes the (cross) entropy. Therefore, the ELBO on $\mathcal{D}_{\text{test}}$ can be higher than the ELBO on $\mathcal{D}_{\text{train}}$, if $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ are not drawn from the same distribution, and $\mathcal{D}_{\text{test}}$ has a lower entropy than $\mathcal{D}_{\text{train}}$. Indeed, this phenomenon has been observed in the out-of-distribution setting when testing on a low-entropy data set (Nalisnick et al., 2018).

Figure 13 shows the ELBOs analogous to the generalization gaps in Figure 2, Figure 10, and Figure 11, but the dotted lines plot the ELBO on the actual training distribution (e.g., on samples from $p_{\text{DM}}(\boldsymbol{x}')$ for DMaaPx). The point of this plot is to warn that comparisons between ELBOs under such different distributions are not meaningful, and should not be used to calculate the generalization gap. For example, note that the plot would suggest a negative generalization gap for data augmentation (purple) on BinaryMNIST. This is consistent with the remark above: since the ELBO is bounded by the negative entropy of the distributions on

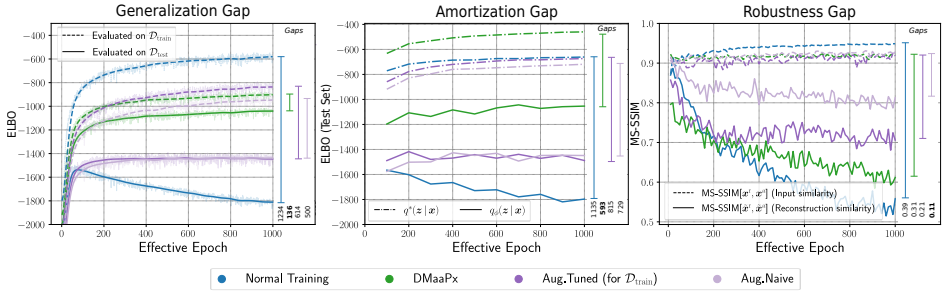


Figure 11: Generalization gap, amortization gap, and robustness gap on the FashionMNIST dataset.

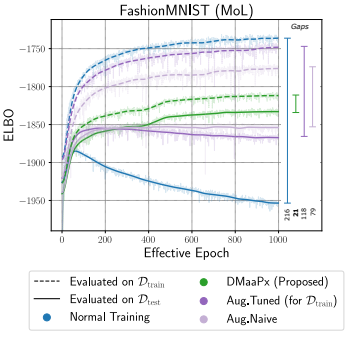


Figure 12: Generalization performance for FashionMINST with MoL likelihood. We observe similar behavior to left panel in Figure 11, which uses a Gaussian likelihood.

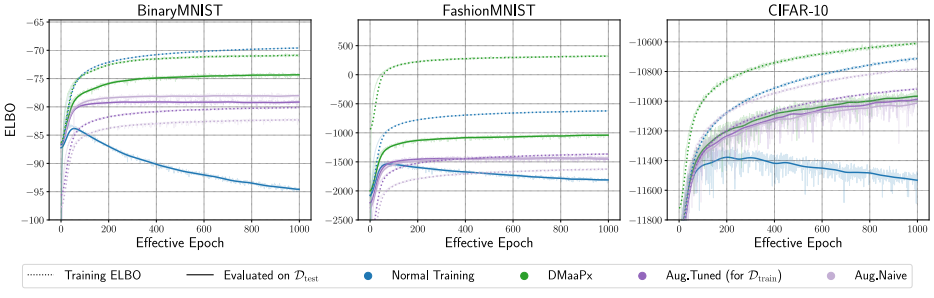


Figure 13: ELBO evaluted on the distribution that is actually used for training (dotted, see Eqs. (7)-(8)). For augmentations, the test ELBO (solid) is higher than the training ELBO (dotted) in the left two panels, which is an artifact of different entropies of the distributions, see Remark.

which it is evaluated, evaluating it on two different distributions with different entropies exhibits differences unrelated to the generalization gap.

B.6 Adding Scheduled Noise to the Training Set

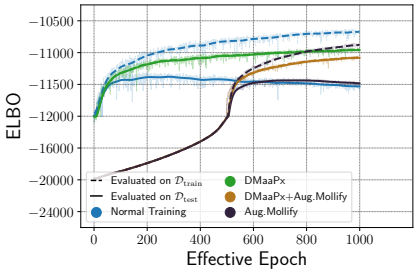


Figure 14: Generalization gap as a function of effective epochs. We compare normal training to DMaaPx, Aug.Mollify, and the combination of DMaaPx and Aug.Mollify. Aug.Mollify reaches a similar performance as the baseline. Combining DMaaPx and Aug.Mollify does not improve upon DMaaPx.

Tran et al. (Tran et al., 2023) show in their recent work that adding (scheduled) noise to the training data during the training process helps the optimization of likelihood-based generative models. More specifically, they anneal the noise schedule from a large variance to a small one for half of the training epochs and, finally, use the original training data for the other half of the training epochs. They argue that data typically resides on a low-dimensional manifold that is embedded in a larger dimension. By adding noise to the data during training one circumvents the problem of density estimation in input space with few samples and, additionally, prevents the model from overfitting to the data manifold. One can also view this idea from a continuation method perspective as the method finds a good initialization point for the optimization.

Figure 14 shows the performance of the mollification method (Aug.Mollify) applied to VAEs on the CIFAR-10 dataset (dark). We compare the performance to normal training (blue), DMaaPx (green), and the combination of DMaaPx and the Aug.Mollify. We find that Aug.Mollify reaches a similar performance as the baseline but does not improve upon it. Also, combining DMaaPx and Aug.Mollify does not bring improvements compared to DMaaPx. However, DMaaPx and Aug.Mollify have similarly small generalization gaps.

B.7 Mixing Synthetic Data with Real Data

We evaluate the performance of VAEs trained on a mix of $x \in \{0, 20, 40, 60, 80, 100\}\%$ of real data and $100 - x\%$ of synthetic data. We train the models without augmentation and with Aug.Tuned. Note that $x = 0\%$ (without augmentation) corresponds to the DMaaPx method and $x = 100\%$ (without augmentation) corresponds to the normal training baseline. $x = 100\%$ (with augmentation) corresponds to the Aug.Tuned baseline.

Considering no augmentation, when moving from a mix of 100% of real data to 0% of real data, we find that the performance is continuously increasing (i.e., the ELBO evaluated on $\mathcal{D}_{\text{test}}$ increases) and the generalization gap is shrinking. Adding augmentation improves both, the ELBO evaluated on $\mathcal{D}_{\text{test}}$ and the generalization gap. Note that DMaaPx ($x = 0\%$, no augmentation) shows the highest ELBO evaluated on $\mathcal{D}_{\text{test}}$ and the smallest generalization gap.

B.8 Training the Diffusion Model on Subsets of the Training Set

We evaluate the perfomance of DMaaPx trained on samples from a diffusion model which has been trained on $x \in \{10, 30, 50, 70, 90\}\%$ of CIFAR-10.

Figure 16 shows ELBO values evaluated on $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. The test performance of DMaaPx using samples from a diffusion model that has been trained on 30% of data equals the test performance of normal training.

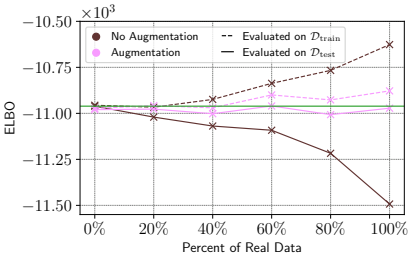


Figure 15: Last ELBO evaluated on $\mathcal{D}_{\text{train}}$ (dashed) and $\mathcal{D}_{\text{test}}$ (solid) with respect to $x\%$ of real data for no augmentation (brown) and Aug.Tuned (pink). $x = 0\%$, brown: DMaaPx. $x = 100\%$, brown: Normal Training. $x = 100\%$, pink: Aug.Tuned. The vertical line shows the ELBO evaluated on the test set for DMaaPx. DMaaPx has highest ELBO and smallest generalization gap. More details are provided in Appendix B.7.

However, DMaaPx shows a significantly smaller generalization gap. Using more data for training the diffusion model improves upon normal training with diminishing returns for $\geq 70\%$ of data.

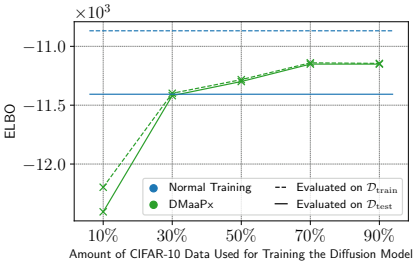


Figure 16: ELBO evaluated on $\mathcal{D}_{\text{train}}$ (dashed) and $\mathcal{D}_{\text{test}}$ (solid) for DMaaPx (green) trained on samples from a diffusion model which has been trained on $x \in \{10, 30, 50, 70, 90\}\%$ of CIFAR-10. We plot the performance of normal training (i.e., VAE trained with the original full CIFAR-10 only) as horizontal blue lines. Using 30% of data for DMaaPx shows similar performance than normal training (with a significantly smaller gap). Using more data outperforms normal training. More details are provided in Appendix B.8.

B.9 Algorithms for Training VAEs with DMaaPx

Below, we provide pseudocode for training VAEs with normal training (Algorithm 1) and DMaaPx (Algorithm 2).

Algorithm 1 Training VAEs - Normal

Given: $\mathcal{D}_{\text{train}}$, $\Theta_0 = \{\theta_0, \phi_0\}$, Number of epochs T, Batch size M

$j = 1$;
for $t = 1, \cdots, T$ **do**
  for $i = 1, \cdots, \frac{|\mathcal{D}_{\text{train}}|}{M}$ **do**
    Get M training examples: $[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M] = \mathcal{D}_{\text{train}}[(i-1) \times M : i \times M]$;
    $\mathcal{L}([\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M]; \Theta_{j-1}) = -\frac{1}{M} \sum_m \left[\text{ELBO}_{\Theta_{j-1}}(\boldsymbol{x}_m)\right]$;
    $\Theta_j = \Theta_{j-1} - \eta \cdot \nabla \mathcal{L}$;
    $j = j + 1$;
  end
end

Algorithm 2 Training VAEs - DMaaPx

Given: $p_{\text{DM}}(\boldsymbol{x})$, $\Theta_0 = \{\theta_0, \phi_0\}$, Number of epochs T, Batch size M, $N = |\mathcal{D}_{\text{train}}|$

// Step 1: Sampling the training data

$\mathcal{D}_{\text{DM}} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T \times N}] \sim p_{\text{DM}}(\boldsymbol{x})$ // Parallelizable; only need to do this once.

// Step 2: Training the VAE

$j = 1$;
for $t = 1, \cdots, T$ **do**
  for $i = 1, \cdots, \frac{N}{M}$ **do**
    Get M training examples: $[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M] = \mathcal{D}_{\text{DM}}[t \times (i-1) \times M : t \times i \times M]$;
    $\mathcal{L}([\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M]; \Theta_{j-1}) = -\frac{1}{M} \sum_m \left[\text{ELBO}_{\Theta_{j-1}}(\boldsymbol{x}_m)\right]$;
    $\Theta_j = \Theta_{j-1} - \eta \cdot \nabla \mathcal{L}$
    $j = j + 1$;
  end
end

C Quantitative Results on Performance Gaps

Table 7 assigns quantitative values to the visual evidence in Figure 2 (CIFAR-10), Figure 11 (FashionMNIST), and Figure 10 (BinaryMNIST).

Table 7: Quantitative values of the performance gaps visualized in Figure 2, Figure 11, and Figure 10. Bold numbers indicate the smallest gap within a dataset.

		generalization gap ($\mathcal{G}_g$, Eq. (3))	amorization gap ($\mathcal{G}_a$, Eq. (4))	robustness gap ($\mathcal{G}_r$, Eq. (5))
Binary MNIST	Normal Training	25.76	20.32	**0.49**
	DMaaPx (ours)	**0.78**	**7.01**	0.50
	Aug.Tuned	8.16	9.34	0.79
	Aug.Naive	6.38	8.16	0.74
Fashion MNIST	Normal Training	1234.50	1135.89	0.39
	DMaaPx (ours)	**136.57**	**593.39**	0.31
	Aug.Tuned	614.93	815.52	0.21
	Aug.Naive	500.33	729.83	**0.11**
CIFAR-10	Normal Training	841.54	835.86	0.41
	DMaaPx (ours)	**5.44**	**288.82**	**0.30**
	Aug.Tuned	94.28	328.08	0.35
	Aug.Naive	228.05	390.25	0.35

D Diffusion Model for DMaaPx

We follow the setup of Karras et al. (Karras et al., 2022) for the design and training of our diffusion model. However, we do not use the proposed augmentation pipeline during training.

We train the diffusion model on 50,000 training data points (i.e., the size of the original CIFAR-10 training set in the case of CIFAR-10) for 4000 epochs. Each model is trained on 8 NVIDIA A100 80GB GPUs for approximately 1 days.

We utilized the deterministic second-order sampler as proposed by Karras et al. (Karras et al., 2022) with 18 integration steps. Each sampled image utilizes a unique initial seed. We sample on a single NVIDIA A100 40GB GPU. Sampling $50,000$ images takes approximately 25 to 30 minutes.

D.1 Samples From the Diffusion Model

Figure 17 shows samples from the diffusion models trained. On CIFAR-10 we report a FID score of 3.9537. Scores on BinaryMNIST and FashionMNIST are ommited as those are not widely reported and heavily depend on preprocessing (Song et al., 2021).

BinaryMNIST FahionMNIST CIFAR-10

Figure 17: Samples of the diffusion models trained on BinaryMNIST (LeCun et al., 1998), FashionMNIST (Xiao et al., 2017), and CIFAR-10 (Krizhevsky et al., 2009).

D.2 Computational Cost of DMaaPx

This section discusses the computational requirements of the DMaaPx method in general.

In this paper, we use the method DMaaPx as a tool to study the difference between naive augmentations and synthetic data from diffusion models when training VAEs. Our evaluations show that there are scenarios where exploiting the modelling assumption (i.e., both VAEs and diffusion models are probabilistic models) helps, which is often overlooked by the generative modelling community.

If a pre-trained diffusion model is available, DMaaPx can be applied with a negligible computational overhead (see Figure 4, right). This setup follows recent trends where using synthetic data from pre-trained diffusion models is becoming a common practice for, e.g., classification (Azizi et al., 2023), robustness (Croce et al., 2021; Wang et al., 2023), and representation learning (Tian et al., 2023).

If no pre-trained diffusion model is available, a diffusion model has to be trained in a first step to apply DMaaPx. Whether this is feasible might depend on the specific application. However, we want to emphasize that once the diffusion model is trained, it can be used to generate as many samples as needed for training as many VAEs as needed.

D.3 Discriminating Synthetic and Real Samples by Classification

One way to evaluate the quality of the generated samples is by training a classifier to discriminate the synthetic and the real data. This method is also known as Classifier Two-Sample Tests (C2ST), which is a metric for evaluating generative models (Bischoff et al., 2024). Here, we train five classifiers to discriminate real data (images taken from CIFAR-10) and synthetic data (images sampled from the diffusion models) where we use $x \in \{30, 50, 70, 90, 100\}\%$ of the original CIFAR-10 to train the diffusion models, respectively.

We use a ResNet 18 (He et al., 2016) for classifying real data (with label 1) and synthetic data (generated by the diffusion model). We train each classifier for 14 epochs with a batch size of 256 with stochastic gradient descent (Robbins & Monro, 1951) with a learning rate of 0.001, a momentum (Rumelhart et al., 1986) of 0.9 and weight decay of 5×10^{-4}.

Figure 18 shows the classification accuracies for discriminating between real and synthetic data. While all samples can be classified with an accuracy $> 50\%$, we find a negative trend between the amount of data that the diffusion model is trained on ($x \in \{30, 50, 70, 90, 100\}\%$) and the classification accuracy. Hence, it is easier to classify samples from a diffusion model that has been trained on less data than samples from a diffusion model that has been trained on more data. Using the maximum amount of training data available, which DMaaPx does, leads to samples that can be distinguished least from real data.

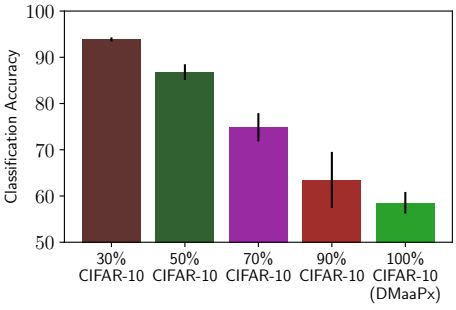


Figure 18: Accuracy of classifier trained to distinguish real data (CIFAR-10) and data sampled from a diffusion model. The diffusion model is trained on $x \in \{30, 50, 70, 90, 100\}\%$ of real data. The more data is used the closer synthetic samples are to real samples (i.e., smaller classification accuracy). Using 100% of CIFAR-10 leads to samples least distinguishable from real data. Accuracies are averaged over 5 random seeds. Error bars show ± 1.96 standard deviations. For details see Appendix D.3.

E Augmentation

We use the augmentation pipeline originally proposed for GAN training following Karras et al. (Karras et al., 2020). Each specific augmentation is applied with a probability of $b \in \{0.1, 0.12\}$. For each dataset we compare two sets of specific augmentations.

1. The hyperparameters for each specific augmentation are tuned by hand with the goal of imitating the data generating distribution that produced the dataset.
2. We use a naive set of specific augmentations that is targeted to image datasets (in general).

Table 8 lists naive augmentation for BinaryMNIST, FashionMNIST, and CIFAR-10. Table 9 lists augmentation tuned to the BinaryMNIST and the FashionMNIST dataset. Table 10 lists augmentation tuned to the CIFAR-10 datset.

They perform similarly overall in Figures 2, 10 and 11. However, Aug.Naive outperforms Aug.Tuned in generalization on BinaryMNIST and FashionMNIST, and in robustness across all datasets. This is surprising as naive augmentation might produce out-of-distribution data, like a horizontally flipped digit "2", potentially impairing performance. Thus, designing augmentation can be labor-intensive.

Table 8: List of specific augmentations applied to BinaryMNIST, FashionMNIST and CIFAR-10. We refer to this set as "naive" augmentation as it is targeted towards images in general (and not towards specific datasets). Each specific augmentation is applied with probability b.

augmentation	description and hyperparameters
horizontal flip	flip an image horizontally
translation	translate an image in x and y direction for $t \in \{0, 1, 2, 3\}$ pixels
scaling	scale an image by $2^{\sigma_{\text{scale}}}$ with $\sigma_{\text{scale}} \in [0, 0.2]$
rotation	rotate an image by d degrees with $d \in [0, 10]$
anisotropic scaling	do anisotropic scaling with scale $2^{\sigma_{\text{aniso-scale}}}$ ($\sigma_{\text{aniso-scale}} \in [0, 0.2]$)
anisotropic rotation	do anisotropic rotation with a probability of 0.5
brightness	change the brightness of an image by $\sigma_{\text{brightness}} \in [0, 0.2]$
contrast	change the contrast of an image by $2^{\sigma_{\text{contrast}}}$ where $\sigma_{\text{contrast}} \in [0, 0.25]$
hue	change the hue by rotation of r_{hue} with $r_{\text{hue}} \in [0, 0.25 \cdot \pi]$
saturation	change the saturation of an image by $2^{\sigma_{\text{saturation}}}$ where $\sigma_{\text{saturation}} \in [0, 0.5]$

Table 9: List of specific augmentations applied to BinaryMNIST and FashionMNIST. The set is tuned towards BinaryMNIST and FashionMNIST. Each specific augmentation is applied with probability b.

augmentation	description and hyperparameters
translation	translate an image in x and y direction for $t \in \{0, 1, 2, 3\}$ pixels
scaling	scale an image by $2^{\sigma_{\text{scale}}}$ with $\sigma_{\text{scale}} \in [0, 0.15]$
rotation	rotate an image by d degrees with $d \in [0, 10]$
anisotropic scaling	do anisotropic scaling with scale $2^{\sigma_{\text{aniso-scale}}}$ ($\sigma_{\text{aniso-scale}} \in [0, 0.15]$)
anisotropic rotation	do anisotropic rotation with a probability of 0.4

Table 10: List of specific augmentations applied to CIFAR-10. The set is tuned towards CIFAR-10. Each specific augmentation is applied with probability b.

augmentation	description and hyperparameters
horizontal flip	flip an image horizontally (applied with probability 1)
vertical flip	flip an image vertically
scaling	scale an image by $2^{\sigma_{\text{scale}}}$ with $\sigma_{\text{scale}} \in [0, 0.2]$
rotation	rotate an image by d degrees with $d \in [0, 360]$
anisotropic scaling	do anisotropic scaling with scale $2^{\sigma_{\text{aniso-scale}}}$ ($\sigma_{\text{aniso-scale}} \in [0, 0.2]$)
anisotropic rotation	do anisotropic rotation with a probability of 0.5

F Practical Evaluation of VAEs on Three Tasks

The improvements of generalization performance, amortized inference and robustness in VAEs have direct impacts on the applications of them. In this section, we evaluate three popular tasks that VAEs are used for, based on whether a task only involves the encoder, the decoder, or both as in (Xiao & Bamler, 2023): (a) representation learning (i.e., using only the encoder); (b) data reconstruction (i.e., using both the encoder and the decoder); and (c) sample generation (i.e., using only the decoder).

Representation learning (with classification as the downstream task). We evaluate the representation learning performance by classification accuracies on the mean $\boldsymbol{\mu}$ of $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ for each $\boldsymbol{x}$. First, we find the learned representations $\boldsymbol{\mu}$ for all data points in the CIFAR-10 test set. Afterwards, we split them into two separate subsets. We use one subset to train the classifier, and test it on the other subset. Our experiments include four different classifiers: logistic regression, a support vector machine (Boser et al., 1992) with radial basis function kernel (SVM-RBF), a SVM with linear kernel (SVM-L), and k-nearest neighbors (kNN) with $k = 5$. Table 11 (representation learning; **RL**) shows the resulting test accuracies across all models considered. We find that VAEs trained with DMaaPx (in bold) are slightly better than other models on average (with overlapping standard deviations), which implies the task of representation learning might benefits from the smaller gaps evaluated in Section 5.2.

Data reconstruction. Tasks such as lossy data compression (Ballé et al., 2017) rely on the reconstruction performance of VAEs. We evaluate the reconstruction performance of VAEs trained on CIFAR-10 using the peak signal-to-noise ratio (PSNR; higher is better). Table 11 (reconstruction; **RC**) shows that DMaaPx performs slightly better than the others on average, but with overlapping standard deviations.

Sample generation. We evaluate the quality of samples generated by VAEs trained on CIFAR-10 with the methods explained in the main text (Normal Training, DMaaPx, Aug.Naive, Aug.Tuned). We report Fréchet Inception Distance (Heusel et al., 2017) (FID; lower is better) and Inception Score (Salimans et al., 2018) (IS; higher is better). Table 11 (sample quality; **SQ**) shows that DMaaPx performs slightly better than the others on average, with overlapping standard deviations, when sample quality is measured in FID, but Normal Training performs better when sample quality is measured in IS.

Overall, VAEs trained with DMaaPx show improvements for representation learning and data reconstruction, and perform similarly to normal training on sample quality. At the same time, VAEs trained with both augmentations seem to have slightly worse performance for representation learning and sample generation, and perform similarly on the reconstruction task when compared to normal training. The results of DMaaPx in the table is consistent with our claim that the proposed method mainly fixes the encoder, which affects representation learning and reconstruction but not sample quality. Additionally, Theis et al. (Theis et al., 2016) show that a generative model with good log-likelihood (i.e., high test ELBO in the case of a VAE) does not necessarily produce great samples.

Table 11: Evaluation of downstream applications of VAEs on CIFAR-10: representation learning with classification as the downstream task (**RL**), reconstruction (**RC**), and sample quality (**SQ**). Results are averaged over 3 random seeds. Note that most differences are smaller than the standard deviations. See Appendix F for a discussion of the results.

		Normal Training	DMaaPx (ours)	Aug.Naive	Aug.Tuned
RL	log. reg. (↑)	0.370 ± 0.018	$\mathbf{0.383 \pm 0.018}$	0.359 ± 0.004	0.361 ± 0.014
	SVM-RBF (↑)	0.427 ± 0.014	$\mathbf{0.438 \pm 0.015}$	0.421 ± 0.004	0.420 ± 0.016
	SVM-L (↑)	0.367 ± 0.015	$\mathbf{0.380 \pm 0.014}$	0.365 ± 0.005	0.366 ± 0.022
	kNN (↑)	0.325 ± 0.006	$\mathbf{0.327 \pm 0.035}$	0.300 ± 0.004	0.299 ± 0.028
RC	PSNR (↑)	16.087 ± 0.042	$\mathbf{16.370 \pm 0.195}$	16.105 ± 0.017	15.924 ± 0.205
SQ	FID (↓)	219.256 ± 16.124	$\mathbf{219.081 \pm 14.894}$	237.238 ± 43.218	240.898 ± 11.072
	IS (↑)	$\mathbf{1.818 \pm 0.155}$	1.614 ± 0.076	1.656 ± 0.047	1.612 ± 0.083

G Practical Evaluation of VAEs on CIFAR-10-C

On CIFAR-10-C (Hendrycks & Dietterich, 2019), we evaluate the representation learning capabilities of VAEs trained with DMaaPx, Aug.Naive, and Aug.Tuned by evaluating the classification performance of a classifier trained in latent space. The experiment are similar to the evaluation of representation learning above (see Appendix F). For each of the 19 classes of corruptions we train four classifiers on a subset of the CIFAR-10-C test set and evaluate its performance on the subset that was not used for training (termed "overall" performance). Additionally, we sample a random (one out of 19 possible) corruption for each of the images of CIFAR-10-C and evaluate its performance. During training we use the strongest degree of corruption.

Figure 19 shows results across all 19 corruptions and the "overall" performance. DMaaPx achieves consistently the best mean across all corruptions for the linear classifiers and DMaaPx achieves the best mean for almost all of the corruptions when regarding the non-linear classifiers. See also Section 5.2 for a discussion of the experiments.

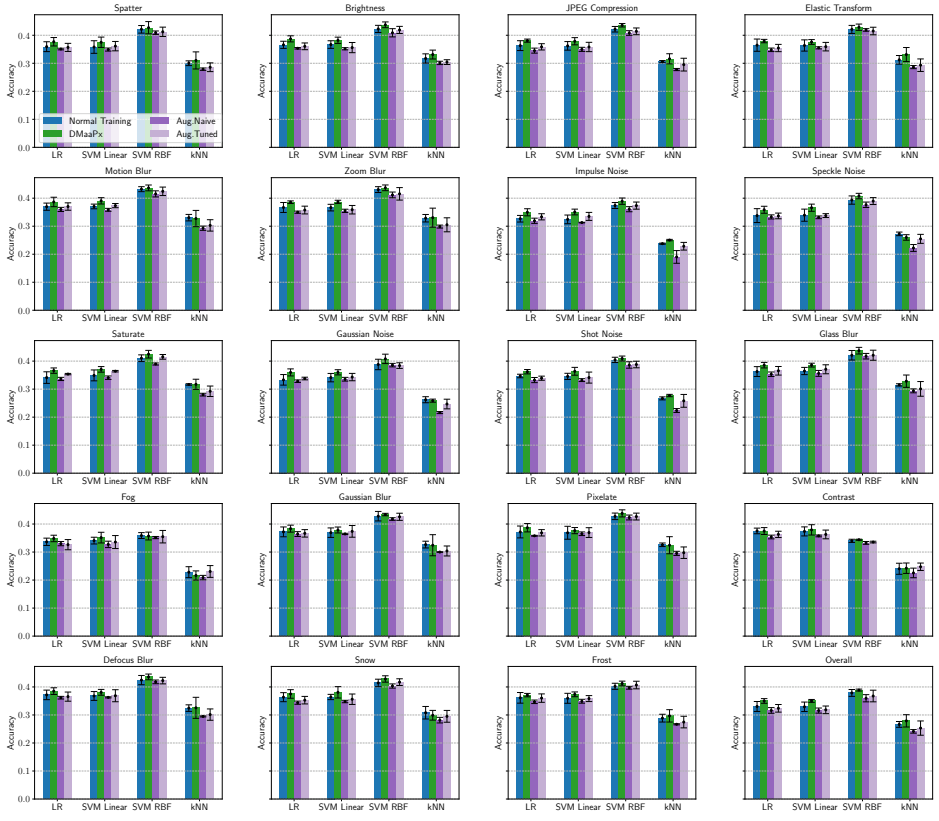


Figure 19: Classification performance of four classifiers (LR: linear regression, SVM kernel: support vector machine with the specified kernel, kNN: k-nearest-neighbor classifier) trained in latent space across the 19 corruptions that are part of CIFAR-10-C. DMaaPx achieves the best mean performance for almost all corruptions under consideration. For details consult Appendix G.

Harvard Data Science Review • Issue 7.3, Summer 2025

Large Language Models Are Zero-Shot Problem Solvers —Just Like Modern Computers

Tim Z. Xiao[1,2] Weiyang Liu[3,2] Robert Bamler[1,2]

[1]University of Tübingen, Tübingen, Baden-Württemberg, Germany,
[2]Max Planck Institute for Intelligent Systems, Tübingen, Baden-Württemberg, Germany,
[3]University of Cambridge, Cambridge, England, United Kingdom

The MIT Press

Published on: Jun 26, 2025

DOI: https://doi.org/10.1162/99608f92.da63c0cb

ABSTRACT

Language models (LMs) are trained specifically for one out of many natural language processing (NLP) tasks. By simply scaling them up to large LMs (LLMs), they emerge the ability to solve many other NLP tasks that they have not been trained for. This emergence of zero-shot problem-solving ability of LLMs surprised the community. In this survey, we draw an overlooked connection between LLMs and modern computers that also emerge zero-shot problem-solving abilities. We discuss their similarities in capability, architecture, and history, and then use these similarities to motivate a different perspective on understanding LLMs and the prompting paradigm. We hope this connection can help us gain a deeper understanding of LLMs, and can spark discussions between the core computer science community and the foundation model community.

Keywords: large language models, universal problem solver, emergence, computer

1. Introduction

Thanks to increasingly powerful computation infrastructures, we have witnessed many significant breakthroughs delivered by simply scaling up deep learning models in recent years. This is partially due to the empirical findings that for many tasks in natural language processing (NLP) and computer vision, the scale of a model (along with the amount of compute and data used for training) and its test performance are so positively correlated that knowing either, one can even predict the other. Such relationship is described by the *scaling laws* (Ganguli et al., 2022; Hestness et al., 2017; Kaplan et al., 2020). In other words, it is not surprising that a larger language model can achieve a better performance in the task of language modeling.

However, recent development in large language models (LLMs) such as GPT-3 (Brown et al., 2020) and ChatGPT (Schulman et al., 2022) has attracted an unusual amount of interest from both the machine learning community as well as the general public. Apart from the imaginative conversations people had with LLMs, a more fundamental reason is that LLMs are not only showing improvements in language modeling but also showing the abilities to solve a wide range of NLP tasks including question answering, translation, summarization, and even algorithmic tasks (Brown et al., 2020; Chowdhery et al., 2022; Radford et al., 2019; Zhou et al., 2022). This means by training on a single task, the model gains the ability to solve tasks it has never been trained for. Such an emergence of *zero-shot*[1] problem-solving abilities in LLMs is surprising and was not predicted from the smaller scale models (Wei et al., 2022). It is still an open question how LLMs emerge with their abilities.

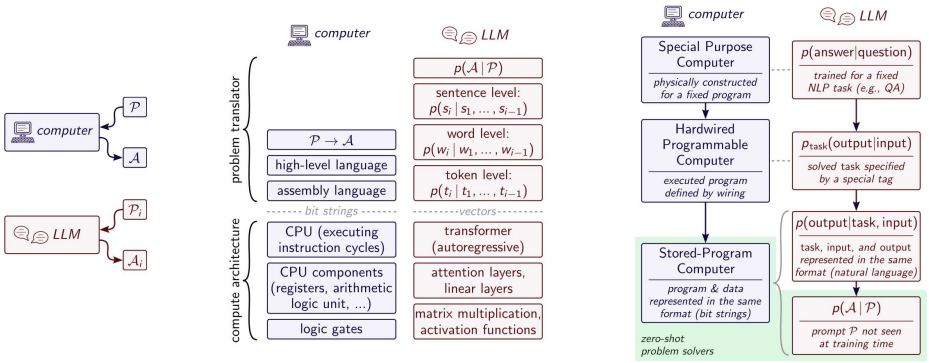
The phenomenon of a system showing spontaneous abilities that cannot be predicted when scaled up is known as *emergence*. Emergence is a phenomenon that is not unique to LLMs. It has been observed and discussed in many other subjects including physics and biology at least half a century ago. In the field of machine learning,

emergence was also discussed and has inspired pioneer works such as the Hopfield network (Hopfield, 1982). The phenomenon is well explicated in an article by Anderson (1972) with a concise title "More Is Different."

Another example that has emergent ability are the computers we use in our daily life. The elementary components in modern computers are logic gates, which are designed to do simple Boolean operations only, for example, *AND*, *OR*. But when we wire a large amount of them together, they emerge with the abilities to run various applications such as video games and text editors. If we phrase problems and tasks into programs, then computers can be seen as zero-shot problem solvers just like LLMs, since most of them are not designed or built for any specific program but as general purpose machines. Crucially, computer scientists have developed a host of tools to help understand and further exploit emergent capabilities, such as debuggers, or various high-level programming paradigms.

In this survey, we connect the dots and draw an overlooked connection between these two zero-shot problem solvers, that is, LLMs and computers. We believe their similarities outline the potential of using LLMs for a much more general problem-solving setting, and their differences help us to understand the behaviors of LLMs and how to improve them. In particular, we illustrate how a good language model, which allows a concept to be encoded less ambiguously, is at the core of exploiting this new model of computation. This LLM-based new computation model has already inspired new computing paradigms. For example, verbalized machine learning (VML; Xiao et al., 2025) introduces the idea of parameterizing machine learning models with natural language, where learning is performed through iterative prompt optimization rather than numerical gradient updates. This bridges traditional machine learning, where models are parameterized numerically and optimized via gradient descent—with a more interpretable, natural language-based paradigm. While this survey focuses on LLMs, a similar story applies to multimodal LLMs (Fei et al., 2022; Li et al., 2024; Liang et al., 2024), which extend the same architecture to other data types by tokenizing inputs like images. We hope that the connections drawn here will spark further discussions on topics such as computation theory, security, distributed computing, and beyond for LLMs.

2. A Tale of Two Zero-Shot Problem Solvers



(A) Framework (B) Levels of abstraction (C) Paths toward universality

Figure 1. Connections between modern computers and large language models from three different perspectives: (a) they both produce corresponding output after doing computations on the input; (b) they both consist of a compute architecture and a problem translator; (c) they both evolved from a specialized task solver to a general zero-shot problem solver.

In this section, we elaborate on the notion that LLMs are zero-shot problem solvers in a similar way in which modern computers are zero-shot problem solvers, since they can both solve much more general problems than those they have been trained or hardcoded for. However, we understand much better how such ability emerged in computers than in LLMs. Here, we look into the similarities as well as the differences in the formulations and the histories of both models of computation, and try to understand how LLMs emerged with their abilities.

Modern Computers. Computers have undergone several transitions from specialized to universal machines (see Figure 1c). As we will discuss in Section 2.1, this transition is similar to the recent history of LLMs. Alan Turing had an early vision (Turing, 1937) of a universal computing machine that was way ahead of its time because the real machines around the time were only constructed for special tasks, such as breaking encrypted communication during wartime. To follow Turing's vision and to avoid building a different machine from scratch for a new task, early computer scientists built machines with a large collection of different arithmetic units and switches (e.g., ENIAC, Weik, 1961). By wiring up these units differently (i.e., programming) and piping the input data through, the machine was able to solve different tasks. While this made it possible to 'reprogram' (i.e., rewire) the computer, the program was still part of the hardware wiring, and any changes to it had to be done manually. The later invention of stored-program computers overcame this limitation and led to even more universal computing machines. Here, the central idea, expressed in the von Neumann architecture (von Neumann, 1945), was to represent programs as data rather than as wiring setups. This allowed computers to read and solve arbitrary programs for which they were not set up by wiring. In other words, they become zero-shot problem solvers.

Large Language Models. Written language can be formed by sequences of tokens. A language model learns mappings to the next token t_i from its prefix. Such a mapping can be represented as a conditional probability distribution (see Figure 1b). The joint probability of the sequence $p(\boldsymbol{t})$ is the product of the conditional probabilities (Bengio et al., 2000), that is, $p(\boldsymbol{t}) = p(t_1)\prod_{i=2}^{N} p(t_i|t_1, \ldots, t_{i-1})$. A popular method to model these conditional probabilities is to use transformers (Vaswani et al., 2017) and train under self-supervision. With the increasing scale of these language models, they turn out to emerge the ability to solve tasks other than language modeling (i.e., they are zero-shot problem solvers), and we call them large language models for distinction.

2.1. Connecting the Dots

Here, we compare the architectures and survey the histories of computers and LLMs, which turned out to be surprisingly similar (see Figure 1). Their likeness might not be a coincidence, and can provide clues for us to understand how LLMs emerge with their abilities.

Similar frameworks. Both computers and LLMs can be seen as mappings from $\mathcal{P}$ to $\mathcal{A}$ (Figure 1a). Given a *program* $\mathcal{P}$, a computer will execute $\mathcal{P}$ and return the answer $\mathcal{A}$ when the program terminates. This mapping is deterministic, which means if $\mathcal{P}$ is a correct implementation of a suitable algorithm, then $\mathcal{A}$ is the correct answer to the problem. Similarly, given a *prompt* $\mathcal{P}$, an LLM will infer and return an answer $\mathcal{A}$. However, this mapping is stochastic and follows the conditional distribution $p(\mathcal{A}|\mathcal{P})$ captured by the LLM. In addition, the interactive feature of LLMs allows us to easily include previous prompts and answers into the new prompt, that is, $\mathcal{P}_i = [\mathcal{P}_{i-1}, \mathcal{A}_{i-1}, new]$.

Similar level of abstractions. Both computers and LLMs consist of a *problem translator* on top of a foundational *compute architecture* (Figure 1b). The problem translator turns problems written in a human readable language into some native representation of the compute architecture (bit strings for computers, vectors for LLMs). The compute architecture is formed by a large collection of simple components (logic gates for computers, matrix multiplications and activation functions for LLMs). In both computers and LLMs, the respective compute architecture is able to do arbitrary computations due to both the scale (i.e., the quantity) and the universality of the involved components.

Similar histories. Both computers and LLMs had a similar path of evolution from task-specific to general purpose problem solvers (Figure 1c). Early approaches to NLP designed and trained a model specifically for a given task, for example, question answering (QA). To create more general solutions, people then experimented with model architectures that were trained on a fixed set of tasks, specified by a task identification tag (Kaiser et al., 2017; Shazeer et al., 2017). Similar to the von Neumann architecture of computers, the next important development was to represent the task description in the same format as the input and output; that is, describing a task in natural language rather than using a special tag (McCann et al., 2018). This resulted in a training objective very similar to language modeling. Inspired by this, Radford et al. (2019) trained a pure (but very

large) language model (i.e., GPT-2) and showed its ability to solve a range of different NLP tasks beyond language modeling.

2.2. Fundamental Difference

While both computers and LLMs can be seen as general purpose problem solvers, they differ fundamentally in what they assume about the nature of problems and how those problems are to be expressed. This difference lies in their epistemological commitments, that is, in what they treat as valid knowledge and representation.

Traditional computers assume a world that is formally structured and logically well-defined. Problems must be specified in a formal language with precise semantics before they can be computed. The act of programming, in this paradigm, is an act of defining the problem space. This worldview aligns with the epistemology of mathematics and formal logic, where truth is absolute and fully specified. As a result, traditional computers are powerful tools for problems that are well-structured, symbolic, and rigorously formalizable.

LLMs, by contrast, are trained on vast corpora of natural language, which embodies ambiguity, cultural context, pragmatic nuance, and incomplete information. They do not assume that a problem must be expressed formally to be computable. Instead, they infer structure and meaning from informal, often underspecified prompts. This reflects a different epistemology, one rooted in interpretation, usage, and statistical regularities in human communication. LLMs are thus adept at solving problems that are difficult to formalize: giving advice, interpreting tone, summarizing arguments, or answering context-dependent questions.

From this perspective, features like ambiguity tolerance and natural language accessibility are not superficial differences between LLMs and computers; they are consequences of deeper commitments about what kinds of problems exist in the world and how those problems are communicated. LLMs allow problems to be posed and solved in the same medium humans use to think and communicate: natural language. Computers demand translation into a symbolic medium, which sacrifices accessibility but gains precision. They differ fundamentally in how they model the world, how they represent problems, and how they reason through solutions. Recognizing this distinction allows us to better understand the types of problems each is best suited to address, and why they complement rather than replace each other.

3. Lessons to Learn

The connections between computers and LLMs identified in Section 2 reveal two important findings about zero-shot problem solvers. Here, we identify two essential building blocks for them, and we point out a direction to improve LLMs, which explains the current focus in the community.

Finding 1. *The zero-shot problem-solving ability only emerges when we combine a powerful compute architecture with a suitable problem translator.*

Computational ability is not as rare a property as we might think. There are many, even simple physical systems that can compute, but zero-shot problem-solving requires an additional layer of abstraction on top of computation that translates new problems into the type of computations that the system can perform. For example, many problems can be phrased as minimization problems, that is, finding the lowest point on some surface. A simple compute architecture capable of solving such an optimization problem could drop a ball onto a physical manifestation of a given surface and record where the ball eventually stops. While, in principle, many problems can be phrased as such minimization problems, in practice, the problems humans want to solve are typically described in a more readable form. Without an additional unit that translates the human-readable description into a surface whose lowest point we want to find, the above computation architecture is not capable of zero-shot problem-solving.

Both computers and LLMs have very powerful compute architectures. Some transformer variants, such as the universal transformer (Dehghani et al., 2019), are theoretically Turing complete, and modern computers are often modeled as Turing machines despite finite resources. However, the relevance of Turing-completeness differs between the two. In classical computing, Turing-completeness is deeply meaningful: it describes a system's ability to simulate any formal, symbolic computation given an appropriate program and sufficient resources. It reflects the system's alignment with a formal epistemology, where problems must be fully specified before they can be solved. LLMs, by contrast, do not require the user to explicitly formalize a problem. Even if they can approximate a Turing machine in theory, their practical power lies in their ability to interpret, generalize, and reason over natural language, even when the input is ambiguous, fuzzy, or pragmatically structured. From this perspective, both systems are computationally capable, but differ in how problems are expressed and reasoned over. What turns both computers and LLMs into zero-shot problem solvers is the presence of powerful problem translators that bridge high-level human intent with the native operations of the compute architecture, preserving and unlocking their full computational potential.

Finding 2. *Minimizing the gap between the language model of LLMs and the internal language model of a user is at the core of unlocking the full problem-solving ability of LLMs.*

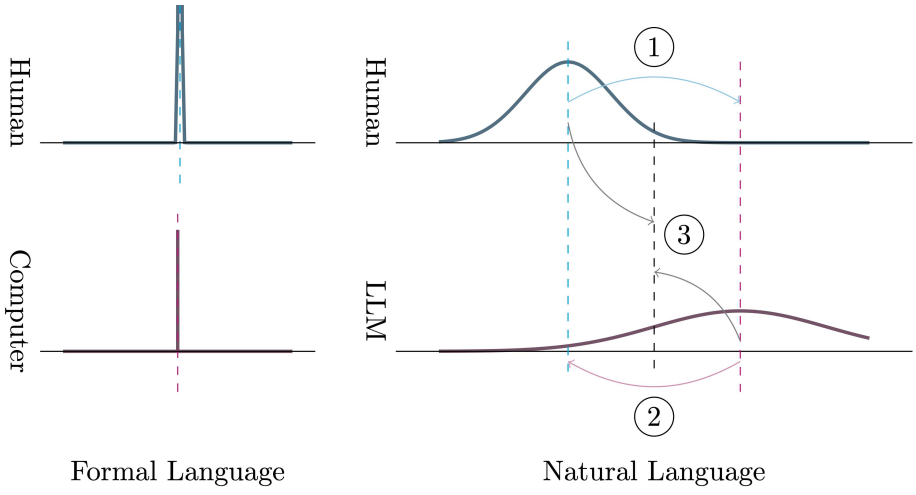


Figure 2. Communication gaps between human and the solver with different languages. Language ambiguity is depicted by variance, model misalignment is depicted by the two separate means

As we have argued above, ensuring that a solver understands the problem or instruction is key to solving it. Computers have been excellent problem solvers for the past decades because formal languages are designed to be precise. This means given a piece of code in some formal language, there is only one correct interpretation (illustrated as a delta distribution in Figure 2 bottom left) and good programmers are expected to understand the language fairly well (see sharp peak in Figure 2 top left). In other words, the models that humans and computers have of a formal language are both relatively *precise*, and the two are closely *aligned* with each other. In contrast, natural language is inherently *imprecise* as there are often many correct interpretations given a sentence. Moreover, both humans and LLMs learn natural language from data rather than from a specification, so their two models will typically be somewhat *misaligned* from each other.

Many of the existing works on LLMs as zero-shot problem solvers can be understood as a spectrum of methods for minimizing the communication gap between humans and LLMs. At one end of the spectrum, we humans can adapt our internal language models toward the language models of LLMs (① in Figure 2), for example, by incorporating certain signaling phrases into our prompts that empirically improve the quality of answers, such as 'Let's think step by step' (Kojima et al., 2022). At the other end of the spectrum, we can adapt LLMs toward our internal language models (② in Figure 2). This can be done during training (e.g., as in InstructGPT and ChatGPT, which were trained with human feedback [Ouyang et al., 2022]), or after training (e.g., personalizing the model by few-shot prompting, i.e., prefixing prompts with examples of similar problems and their solutions or with a conversation history). Most prompting research falls somewhere in

between these two extremes, where humans and LLMs adapt their models toward each other (③ in Figure 2) (Liu et al., 2023).

Recent reinforcement-learning-tuned models such as DeepSeek-R1 (Guo et al., 2025) and OpenAI's o1 (Jaech et al., 2024) can also be seen as better aligned to the user's internal language model. While they remain within the same computational paradigm, they exhibit stronger reasoning and interaction capabilities through reinforcement learning. Analogous to different instruction set architectures (e.g., ARM vs. x86) or to built-in runtime systems with enhanced optimizations and debugging, they represent more capable implementations of the same general purpose solver.

However, as LLMs become more aligned with users and easier to interact with, concerns about overreliance and cognitive off-loading are increasingly relevant. Emerging research shows that while LLMs can enhance productivity and creativity, frequent dependence may lead to reduced critical thinking, skill degradation, and even psychological dependence (Gerlich, 2025; Huang et al., 2024; Kumar et al., 2025; Macnamara et al., 2024; Sabbah & Li, 2025). These risks suggest that narrowing the communication gap between humans and LLMs must be balanced with design considerations that preserve human agency, oversight, and long-term cognitive resilience.

4. Imagining a Natural Language Computing Paradigm

The previous sections highlighted how LLMs, like early computers, have evolved from task-specific to general purpose problem solvers. If this trajectory continues, natural language systems may face challenges similar to those encountered in the history of computing. This section explores how such systems might scale, what principles could guide their development, and what new subdomains may emerge as a result.

4.1. Integrating Algorithms With Natural Language-Based Programs

Prompts can be viewed as programs written in natural language, with LLMs acting as interpreters over a distributional semantic space. While less formally specified than code, prompts define tasks, condition inputs, and elicit structured behaviors, ranging from deterministic functions (e.g., sorting a list) to abstract reasoning (e.g., summarizing an argument).

Reframing prompting as computation opens the door to adapting classical algorithmic techniques to this new medium. In particular, deterministic algorithms can be combined with natural language–defined components. For example, the VML framework (Xiao et al., 2025) mentioned in the introduction retains the structure of traditional learning algorithms while expressing models and optimizers in natural language, executed via LLMs. Prompt design, in this view, becomes an instance of program synthesis, where the goal is to construct language instructions that elicit desired model behavior reliably and efficiently. This hybrid view positions LLMs as semantic engines integrated into algorithmic frameworks, enabling procedural reasoning over language.

4.2. Modularity and Encapsulation for Scalable Reasoning

A key factor in the scalability of classical computing has been modularization: the ability to build complex systems from smaller, reusable components with well-defined interfaces. The same principle will be essential for scaling natural language–based computation. Here, prompt modules act as encapsulated language functions, each performing a specific subtask such as summarization, classification, or verification.

By defining clear input-output formats (e.g., a paragraph, a Boolean answer, a list) through natural language instructions, these modules serve as application programming interfaces (APIs). This enables the decomposition of complex tasks into sequences of simpler ones, where each module can be tested, reused, and improved independently. Encapsulation ensures that internal reasoning remains isolated, allowing modules to interact through well-specified linguistic contracts without entangling their internal logic.

As in classical systems, such modular, interface-driven design will be essential for scaling LLM-based systems, supporting composability, robustness, and maintainability.

4.3. Language-Native Infrastructure and Emerging Subdomains

Scalable natural language computation will require infrastructure analogous to classical computing stacks, adapted to a linguistic interface. Orchestration frameworks will manage memory, dialogue context, tool usage, and agent coordination. Prompt compilers may translate high-level intent into structured prompt templates, resolving ambiguity and optimizing for task-specific performance. Execution layers will track progress, handle branching and fallback strategies, and ensure semantic consistency over multiturn or multiagent workflows.

As this infrastructure matures, we can anticipate the emergence of subdomains structurally similar to those in classical computer science. Security, for example, will involve defending against prompt injection, adversarial language, and unintended emergent behaviors. Distributed systems will manifest as multiagent collaborations, coordinated through dialogue protocols and turn-taking conventions. Memory and retrieval will rely on hybrid systems that combine embedding-based indexing with contextual synthesis. Networking will involve language-based protocols for structured message exchange among LLMs, tools, and users. These subdomains will not replicate their classical counterparts directly, but they will arise from similar demands for reliability, coordination, and scalability.

4.4. Toward a Theory of Language-Based Computation

The development of natural language computing calls for a new theoretical foundation. Unlike classical computation, which is grounded in formal logic and deterministic semantics, language-based computation is probabilistic, context-dependent, and informally specified. A theory in this space should help us understand which tasks can be solved effectively through prompts, and which are better addressed by traditional methods.

Such a theory may draw on both formal reasoning and empirical benchmarking, with insights often conditioned on model performance over specific tasks. In some cases, the benchmark itself may form an integral part of the theory; that is, defining the context under which certain behaviors or guarantees are expected to hold.

It should also account for the cost of formalization, a factor often overlooked in classical theory. LLMs offer practical value by operating directly on informal inputs, bypassing the need for fully specified representations. Capturing this advantage is essential for modeling the real-world strengths of language-based systems.

Ultimately, such a theory should offer insight into the kinds of problems LLMs are well-suited to solve, and under what conditions. This understanding would help guide the design of scalable reasoning algorithms that make effective use of language-based computation.

Disclosure Statement

The authors have no financial or nonfinancial disclosures to share for this article.

References

Anderson, P. W. (1972). More is different. *Science*, *177*(4047), 393–396. https://doi.org/10.1126/science.177.4047.393

Bengio, Y., Ducharme, R., & Vincent, P. (2000). A neural probabilistic language model. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems* (Vol. 13, pp. 932–938). Curran Associates. https://papers.nips.cc/paper_files/paper/2000/hash/728f206c2a01bf572b5940d7d9a8fa4c-Abstract.html

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, 1877–1901). Curran Associates. https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., . . . Fiedel, N. (2022). *PaLM: Scaling language modeling with pathways*. ArXiv. https://doi.org/10.48550/arXiv.2204.02311

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2019, May 6–9). *Universal transformers* [Presentation]. ICLR 2019: Proceedings of the 7th international conference on learning representations, New Orleans, LA, United States. https://openreview.net/forum?id=HyzdRiR9Y7

Fei, N., Lu, Z., Gao, Y., Yang, G., Huo, Y., Wen, J., Lu, H., Song, R., Gao, X., Xiang, T., Sun, H., & Wen, J. (2022). Towards artificial general intelligence via a multimodal foundation model. *Nature Communications*, *13*(1), Article 3094. https://doi.org/10.1038/s41467-022-30761-2

Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., El Showk, S., Fort, S., Hatfield-Dodds, Z., Henighan, T., Johnston, S., Jones, A., Joseph, N., Kernian, J., Kravec, S., . . . Clark, J. (2022). Predictability and surprise in large generative models. In *Proceedings of 2022 ACM conference on fairness, accountability, and transparency* (pp. 1747–1764). Association for Computing Machinery. https://doi.org/10.1145/3531146.3533229

Gerlich, M. (2025). AI tools in society: Impacts on cognitive offloading and the future of critical thinking. *Societies*, *15*(1), Article 6. https://doi.org/10.3390/soc15010006

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., . . . Zhang, Z. (2025). *Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning*. ArXiv. https://doi.org/10.48550/arXiv.2501.12948

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M., Ali, M., Yang, Y., & Zhou, Y. (2017). *Deep learning scaling is predictable, empirically*. ArXiv. https://doi.org/10.48550/arXiv.1712.00409

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558. https://doi.org/10.1073/pnas.79.8.2554

Huang, S., Lai, X., Ke, L., Li, Y., Wang, H., Zhao, X., Dai, X., & Wang, Y. (2024). AI technology panic—is AI dependence bad for mental health? A cross-lagged panel model and the mediating roles of motivations for AI use among adolescents. *Psychology Research and Behavior Management*, *17*, 1087–1102. https://doi.org/10.2147/PRBM.S440889

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., . . . Li, Z. (2024). *OpenAI o1 system card*. ArXiv. https://doi.org/10.48550/arXiv.2412.16720

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., & Uszkoreit, J. (2017). *One model to learn them all*. ArXiv. https://doi.org/10.48550/arXiv.1706.05137

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling laws for neural language models*. ArXiv. https://doi.org/10.48550/arXiv.2001.08361

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Proceedings of the 36th international conference on advances in neural information processing systems* (Article 1613). Curran Associates. https://proceedings.neurips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html

Kumar, H., Vincentius, J., Jordan, E., & Anderson, A. (2025). Human creativity in the age of LLMs: Randomized experiments on divergent and convergent thinking. In N. Yamashita, V. Evers, K. Yatani, S. X. Ding, B. Lee, M. Chetty, & P. O. T. Dugas (Eds.), *Proceedings of the 2025 CHI conference on human factors in computing systems* (Article 23). Association for Computing Machinery. https://doi.org/10.1145/3706598.3714198

Li, C., Gan, Z., Yang, Z., Yang, J., Li, L., Wang, L., & Gao, J. (2024). Multimodal foundation models: From specialists to general-purpose assistants. *Foundations and Trends® in Computer Graphics and Vision*, *16*(1-2). https://doi.org/10.1561/0600000110

Liang, P. P., Zadeh, A., & Morency, L.-P. (2024). Foundations & trends in multimodal machine learning: Principles, challenges, and open questions. *ACM Computing Surveys*, *56*(10), Article 264. https://doi.org/10.1145/3656580

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, *55*(9), Article 195. https://doi.org/10.1145/3560815

Macnamara, B. N., Berber, I., Çavuşoğlu, M. C., Krupinski, E. A., Nallapareddy, N., Nelson, N. E., Smith, P. J., Wilson-Delfosse, A. L., & Ray, S. (2024). Does using artificial intelligence assistance accelerate skill decay and hinder skill development without performers' awareness? *Cognitive Research: Principles and Implications*, *9*(1), Article 46. https://doi.org/10.1186/s41235-024-00572-8

McCann, B., Keskar, N. S., Xiong, C., & Socher, R. (2018). *The natural language decathlon: Multitask learning as question answering*. ArXiv. https://doi.org/10.48550/arXiv.1806.08730

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., & Lowe, R. (2022). Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information*

processing systems (Vol. 35, pp. 27730–27744). Curran Associates. https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners*. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Sabbah, J., & Li, F. (2025). When humans and large language models collaborate, problem-finding illuminates. *Innovation*. https://doi.org/10.1080/14479338.2025.2504428

Schulman, J., Zoph, B., Kim, C., Hilton, J., Menick, J., Weng, J., Uribe, J. F. C., Fedus, L., Metz, L., Pokorny, M., Lopes, R. G., Zhao, S., Vijayvergiya, A., Sigler, E., Perelman, A., Voss, C., Heaton, M., Parish, J., Cummings, D., . . . Ryder, N. (2022, November 30). ChatGPT: Optimizing language models for dialogue. *OpenAI*. https://openai.com/index/chatgpt/

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., & Dean, J. (2017, April 24 - 26). *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer* [Poster presentation]. ICLR 2017: Proceedings of the 5th International Conference on Learning Representations, Toulon, France. https://openreview.net/forum?id=B1ckMDqlg

Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), Article 230. https://doi.org/10.1112/plms/s2-42.1.230

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30, 5998–6008). Curran Associates. https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

von Neumann, J. (1945). *First draft of a report on the EDVAC*. University of Pennsylvania. https://library.si.edu/digital-library/book/firstdraftofrepo00vonn

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=yzkSU5zdwD

Weik, M. H. (1961). The ENIAC story. *Ordnance*, *45*(244), 571–575.

Xiao, T. Z., Bamler, R., Schölkopf, B., & Liu, W. (2025). Verbalized machine learning: Revisiting machine learning with language models. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=k3Ab6RuJE9

Zhou, H., Nova, A., Larochelle, H., Courville, A., Neyshabur, B., & Sedghi, H. (2022). *Teaching algorithmic reasoning via in-context learning*. ArXiv. https://doi.org/10.48550/arXiv.2211.09066

Footnotes

1. We use the term *zero-shot* to highlight that the model is not trained for the tested tasks (similar to its usage in prior work [Radford et al., 2019]). This is sometimes also called *universal* or *general* problem-solving. In many other LLM papers (Kojima et al., 2022), *zero-shot* means no demonstration is provided during inference time in the prompt. Here we treat both zero-shot and few-shot prompting as examples of zero-shot problem solving. ↩

References

- Anderson, P. W. (1972). More is different. *Science*, *177*(4047), 393–396. https://doi.org/10.1126/science.177.4047.393 ↩
- Bengio, Y., Ducharme, R., & Vincent, P. (2000). A neural probabilistic language model. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems* (Vol. 13, pp. 932–938). Curran Associates. https://papers.nips.cc/paper_files/paper/2000/hash/728f206c2a01bf572b5940d7d9a8fa4c-Abstract.html ↩
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, & H.-T. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 1877–1901). Curran Associates. https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html ↩
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., … Fiedel, N. (2022). *PaLM: Scaling language modeling with pathways*. ArXiv. https://doi.org/10.48550/arXiv.2204.02311 ↩
- Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2019, May 6–9). *Universal transformers* [Presentation]. ICLR 2019: Proceedings of the 7th international conference on learning representations, New

↩

- Fei, N., Lu, Z., Gao, Y., Yang, G., Huo, Y., Wen, J., Lu, H., Song, R., Gao, X., Xiang, T., Sun, H., & Wen, J.-R. (2022). Towards artificial general intelligence via a multimodal foundation model. *Nature Communications*, *13*(1), Article 3094. https://doi.org/10.1038/s41467-022-30761-2 ↩
- Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., El Showk, S., Fort, S., Hatfield-Dodds, Z., Henighan, T., Johnston, S., Jones, A., Joseph, N., Kernian, J., Kravec, S., … Clark, J. (2022). Predictability and surprise in large generative models. In *Proceedings of 2022 ACM conference on fairness, accountability, and transparency* (pp. 1747–1764). Association for Computing Machinery. https://doi.org/10.1145/3531146.3533229 ↩
- Gerlich, M. (2025). AI tools in society: Impacts on cognitive offloading and the future of critical thinking. *Societies*, *15*(1), Article 6. https://doi.org/10.3390/soc15010006 ↩
- Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., … Zhang, Z. (2025). *Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning*. ArXiv. https://doi.org/10.48550/arXiv.2501.12948 ↩
- Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M., Ali, M., Yang, Y., & Zhou, Y. (2017). *Deep learning scaling is predictable, empirically*. ArXiv. https://doi.org/10.48550/arXiv.1712.00409 ↩
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558. https://doi.org/10.1073/pnas.79.8.2554 ↩
- Huang, S., Lai, X., Ke, L., Li, Y., Wang, H., Zhao, X., Dai, X., & Wang, Y. (2024). AI technology panic—is AI dependence bad for mental health? A cross-lagged panel model and the mediating roles of motivations for AI use among adolescents. *Psychology Research and Behavior Management*, *17*, 1087–1102. https://doi.org/10.2147/PRBM.S440889 ↩
- Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., … Li, Z. (2024). *OpenAI o1 system card*. ArXiv. https://doi.org/10.48550/arXiv.2412.16720 ↩
- Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., & Uszkoreit, J. (2017). *One model to learn them all*. ArXiv. https://doi.org/10.48550/arXiv.1706.05137 ↩
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling laws for neural language models*. ArXiv. https://doi.org/10.48550/arXiv.2001.08361 ↩
- Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Proceedings of the 36th international conference on advances in neural information processing systems* (p. Article 1613). Curran Associates.

https://proceedings.neurips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html↩

- Kumar, H., Vincentius, J., Jordan, E., & Anderson, A. (2025). Human creativity in the age of LLMs: Randomized experiments on divergent and convergent thinking. In N. Yamashita, V. Evers, K. Yatani, S. X. Ding, B. Lee, M. Chetty, & P. O. T. Dugas (Eds.), *Proceedings of the 2025 CHI conference on human factors in computing systems* (p. Article 23). Association for Computing Machinery. https://doi.org/10.1145/3706598.3714198 ↩
- Li, C., Gan, Z., Yang, Z., Yang, J., Li, L., Wang, L., & Gao, J. (2024). Multimodal foundation models: From specialists to general-purpose assistants. *Foundations and Trends® in Computer Graphics and Vision*, *16*(1–2). https://doi.org/10.1561/0600000110 ↩
- Liang, P. P., Zadeh, A., & Morency, L.-P. (2024). Foundations & trends in multimodal machine learning: Principles, challenges, and open questions. *ACM Computing Surveys*, *56*(10), Article 264. https://doi.org/10.1145/3656580 ↩
- Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, *55*(9), Article 195. https://doi.org/10.1145/3560815 ↩
- Macnamara, B. N., Berber, I., Çavuşoğlu, M. C., Krupinski, E. A., Nallapareddy, N., Nelson, N. E., Smith, P. J., Wilson-Delfosse, A. L., & Ray, S. (2024). Does using artificial intelligence assistance accelerate skill decay and hinder skill development without performers' awareness? *Cognitive Research: Principles and Implications*, *9*(1), Article 46. https://doi.org/10.1186/s41235-024-00572-8 ↩
- McCann, B., Keskar, N. S., Xiong, C., & Socher, R. (2018). *The natural language decathlon: Multitask learning as question answering*. ArXiv. https://doi.org/10.48550/arXiv.1806.08730 ↩
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., & Lowe, R. (2022). Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (Vol. 35, pp. 27730–27744). Curran Associates. https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html ↩
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf ↩
- Sabbah, J., & Li, F. (2025). When humans and large language models collaborate, problem-finding illuminates. *Innovation*. https://doi.org/10.1080/14479338.2025.2504428 ↩
- Schulman, J., Zoph, B., Kim, C., Hilton, J., Menick, J., Weng, J., Uribe, J. F. C., Fedus, L., Metz, L., Pokorny, M., Lopes, R. G., Zhao, S., Vijayvergiya, A., Sigler, E., Perelman, A., Voss, C., Heaton, M., Parish,

J., Cummings, D., … Ryder, N. (2022, November 30). ChatGPT: Optimizing language models for dialogue. *OpenAI*. https://openai.com/index/chatgpt/↩

- Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., & Dean, J. (2017, April 24 - 26). *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer* [Poster presentation]. ICLR 2017: Proceedings of the 5th International Conference on Learning Representations, Toulon, France. https://openreview.net/forum?id=B1ckMDqlg

  ↩
- Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), Article 230. https://doi.org/10.1112/plms/s2-42.1.23 ↩
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30, pp. 5998–6008). Curran Associates. https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html ↩
- von Neumann, J. (1945). *First draft of a report on the EDVAC*. University of Pennsylvania. https://library.si.edu/digital-library/book/firstdraftofrepo00vonn

  ↩
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=yzkSU5zdwD ↩
- Weik, M. H. (1961). The ENIAC story. *Ordnance*, *45*(244), 571–575. ↩
- Xiao, T. Z., Bamler, R., Schölkopf, B., & Liu, W. (2025). Verbalized machine learning: Revisiting machine learning with language models. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=k3Ab6RuJE9 ↩
- Zhou, H., Nova, A., Larochelle, H., Courville, A., Neyshabur, B., & Sedghi, H. (2022). *Teaching algorithmic reasoning via in-context learning*. ArXiv. https://doi.org/10.48550/arXiv.2211.09066 ↩

Published in Transactions on Machine Learning Research (01/2025)

Verbalized Machine Learning: Revisiting Machine Learning with Language Models

Tim Z. Xiao *zhenzhong.xiao@uni-tuebingen.de*
Max Planck Institute for Intelligent Systems, Tübingen & University of Tübingen

Robert Bamler *robert.bamler@uni-tuebingen.de*
University of Tübingen

Bernhard Schölkopf *bernhard.schoelkopf@tuebingen.mpg.de*
Max Planck Institute for Intelligent Systems, Tübingen

Weiyang Liu* *weiyang.liu@tuebingen.mpg.de*
Max Planck Institute for Intelligent Systems, Tübingen & University of Cambridge

Reviewed on OpenReview: *https://openreview.net/forum?id=k3Ab6RuJE9*

Abstract

Motivated by the progress of large language models (LLMs), we introduce the framework of verbalized machine learning (VML). In contrast to conventional machine learning (ML) models that are typically optimized over a continuous parameter space, VML constrains the parameter space to be human-interpretable natural language. Such a constraint leads to a new perspective of function approximation, where an LLM with a text prompt can be viewed as a function parameterized by the text prompt. Guided by this perspective, we revisit classical ML problems, such as regression and classification, and find that these problems can be solved by an LLM-parameterized learner and optimizer. The major advantages of VML include (1) easy encoding of inductive bias: prior knowledge about the problem and hypothesis class can be encoded in natural language and fed into the LLM-parameterized learner; (2) automatic model class selection: the optimizer can automatically select a model class based on data and verbalized prior knowledge, and it can update the model class during training; and (3) interpretable learner updates: the LLM-parameterized optimizer can provide explanations for why an update is performed. We empirically verify the effectiveness of VML, and hope that VML can serve as a stepping stone to stronger interpretability.

"The limits of my language mean the limits of my world."

— Ludwig Wittgenstein

1 Introduction

The unprecedented success of large language models (LLMs) has changed the way people solve new problems in machine learning. Compared to conventional end-to-end training where a neural network is trained from scratch on some curated dataset, it has become increasingly more popular to leverage a pretrained LLM and design good prompts that contain in-context examples and effective instructions. These two ways of problem-solving lead to an intriguing comparison. Traditionally, we would optimize a neural network in *a continuous numerical space* using gradient descent, while in the new approach, we optimize the input prompt of an LLM in *a discrete natural language space*. Since a neural network is effectively a function parameterized by its numerical weights, can a pretrained LLM act as a function parameterized by its text prompt?

Driven by this question, we conceptualize the framework of verbalized machine learning (VML), which uses natural language as the representation of the model parameter space. *The core idea behind VML is that we*

*Corresponding author

can define a machine learning model using natural language, and the training of such a model is based on the iterative update of natural language. This framework enables many new possibilities for interpretability, as the decision rules and patterns learned from data are stored and summarized in natural language. Specifically, we propose to view the input text prompt of LLMs as the model parameters that are being learned. However, optimization over such a natural language parameter space also introduces additional difficulties. Inspired by previous work [5, 25] where the optimizer is viewed as a function parameterized by a neural network, we parameterize the optimizer function as another LLM, which produces the next-step model parameters by taking in the current model parameters, a batch of training data points, and the loss function. Therefore, VML requires the optimizer LLM to update the learner LLM iteratively towards the training objective.

Compared to conventional numerical machine learning, the VML framework brings a few unique advantages. First, VML introduces an easy and unified way to encode inductive bias into the model. Because the model parameters are fully characterized by human-interpretable natural language, one can easily enter the inductive bias using language. This linguistic parameterization makes machine learning models fully interpretable and adjustable. For example, if the input and output data are observed to be linearly correlated, then one can use this sentence as part of text prompt. How to effectively encode inductive bias is actually a longstanding problem in machine learning, and VML provides a unified way to inject the inductive bias through natural language—just like teaching a human learner. Second, VML performs automatic model selection during the learning process. The optimizer LLM can automatically select a suitable model class based on the training data and verbalized prior knowledge. Third, each update of the model is fully interpretable in the sense that the optimizer LLM can give an explanation of why it chooses such an update. One can even interact with the optimizer LLM in order to inject new prior knowledge or obtain detailed reasoning.

VML can be viewed as a natural generalization of in-context learning (ICL). Specifically, ICL is a single-step implicit learning process, while VML is a multi-step iterative learning process where the in-context examples are summarized into verbal pattern and knowledge. Moreover, VML offers a sequential (or conditional) way for scaling inference-time compute [7, 48]. Compared to the best-of-N re-sampling, VML iteratively updates its model parameter prompt by taking into account the learner's past predictions.

An important concept of VML is its unified token-level representation of both data and model. Unlike numerical machine learning, language models in VML do not differentiate data and model, and treat both of them as part of the text prompt. This shares a striking connection to stored-program computers, also known as the von Neumann architecture, where the key idea is to represent programs as data rather than wiring setups. The link between language models and stored-program computers underscores the importance of text prompts, which play a similar role to computer programs, and, along with LLMs, can become a powerful zero-shot problem solver. Our contributions are as follows:

- We formulate the framework of verbalized machine learning, where pretrained language models are viewed as function approximators parameterized by their text prompts. Then, we revisit a few simple machine learning problems and show that VML is able to solve them.
- We design a concrete VML algorithm with a text prompt template. This algorithm parameterizes both the learner model and the optimizer as LLMs, and enables the iterative verbalized training.
- We conduct empirical studies for the injection of verbalized inductive bias and show that it is promising to use natural language as a unified way to encode prior knowledge. Moreover, we validate the effectiveness of VML in different applications (Section 4, Appendix A, E,F,G,H).

2 Related Work

LLMs for planning and optimization. Language models are used to perform planning for embodied agents [49, 60, 26, 28], such that they can follow natural language instruction to complete complex tasks. More recently, LLMs have been used to solve optimization problems [63]. Specifically, the LLM generates a new solution to an optimization problem from a prompt that contains previously generated solutions and their loss values. The LLM optimizer in [63] shares a high-level similarity to our work, as we also aim to solve an optimization problem with LLMs. The key difference to [63] is our function approximation view of LLMs, which enables us to revisit classical machine learning problems and solve them in the VML framework.

Natural language to facilitate learning. [45, 23, 24, 37, 71] show that natural language captions serve as an effective supervision to learn transferable visual representation. [35, 38, 40, 34, 66, 61] find that natural language descriptions can easily be turned into zero-shot classification criteria for images. [4] proposes to use natural language as latent parameters to characterize different tasks in few-shot learning. In contrast, VML uses the text prompt of LLMs to parameterize functions and learns this prompt in a data-driven fashion.

Prompt engineering and optimization. There are many prompting methods [56, 72, 73, 54, 67, 68, 58] designed to elicit the reasoning ability of LLMs. To reduce the efforts in designing good prompts, prompt optimization [72, 73, 63, 41, 57, 11, 27, 32, 50, 70] has been proposed. VML can be viewed as a special instance of prompt optimization, but unlike many current generic prompt optimization methods that search the optimal text prompts through best-of-N sampling without reasoning, VML updates its text-based parameters by explicitly reasoning about the incorrect predictions and learning the underlying data pattern based on the reasoning outcome, which ensures that the learner in VML remains fully interpretable. To summarize, the difference between VML and current generic prompt optimization (*e.g.*, [73]) is similar to the difference between *gradient-based* and *gradient-free* optimization. Another subtle difference that separates the two is that the goal of VML (like classical machine learning) is to learn a model to recognize a generalizable pattern in a given training set, while the goal of current prompt optimization (like classical optimization) is more general, which is to simply optimize an objective function (without the need to learn the underlying pattern, *e.g.*, [41]). We can phrase the goal of a machine learning problem into an optimization objective, and use the tools from optimization to solve it, but the focus of the two areas are fundamentally different. We compare the difference of the two in the experiments (see Section 4.8 and Appendix D). We observe that current prompt optimization often results in a generic instruction rather than a description of the data pattern.

LLMs for multi-agent systems. Due to the strong instruction-following ability, LLMs are capable of playing different roles in a multi-agent systems. [42, 59, 15, 22] study a multi-agent collaboration system for solving complex tasks like software development. VML can also be viewed as a two-agent system where one LLM plays the learner role and the other LLM plays the optimizer role.

3 Verbalized Machine Learning

3.1 From Numerical to Verbalized Machine Learning

Classical machine learning models (*e.g.*, neural networks) are typically trained within a continuous numerical parameter space. Once trained, these models are stored as a collection of numerical values that are not interpretable and remain a black box. Motivated by the strong universal problem-solving capability of LLMs, we find it appealing to view an LLM as a function approximator that is parameterized by its own text prompt. This perspective leads to our VML framework. Similar to a general-purpose modern computer whose functionality is defined by its running program, a function that is defined by an LLM is characterized by its text prompt. Due to the fully human-interpretable text prompt, the VML framework provides strong interpretability for its learned function and is also easy to trace the cause of model failure. Figure 1 gives a comparison between numerical machine learning and VML. In the proposed VML framework, both data and model are represented in a unified token-based format, while numerical machine learning treats data and model differently.

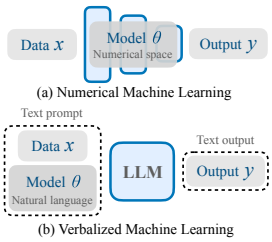


Figure 1: A comparison between numerical machine learning and VML.

3.2 Natural Language as the Model Parameter Space

VML parameterizes a machine-learning model with natural language. More formally, VML places a strong constraint on the model parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, \cdots, \theta_t\} \in \Theta_{\text{language}}$ to exchange for interpretability, where $\boldsymbol{\theta}$ is a text token sequence, $\theta_t \in \mathcal{A}, \forall t$ is some text token from a large token set $\mathcal{A}$, and Θ_{language} denotes the set of all natural language sequences that humans can understand. The model parameter space in VML has the following properties: (1) discrete: the natural language space is discrete; (2) sequential: the natural language space is sequential, and the next word is dependent on its previous words. In contrast, the parameter space in numerical machine learning is not sequentially dependent; and (3) human-interpretable: the natural language that characterizes the model is human-interpretable. More discussions are given in Appendix J.

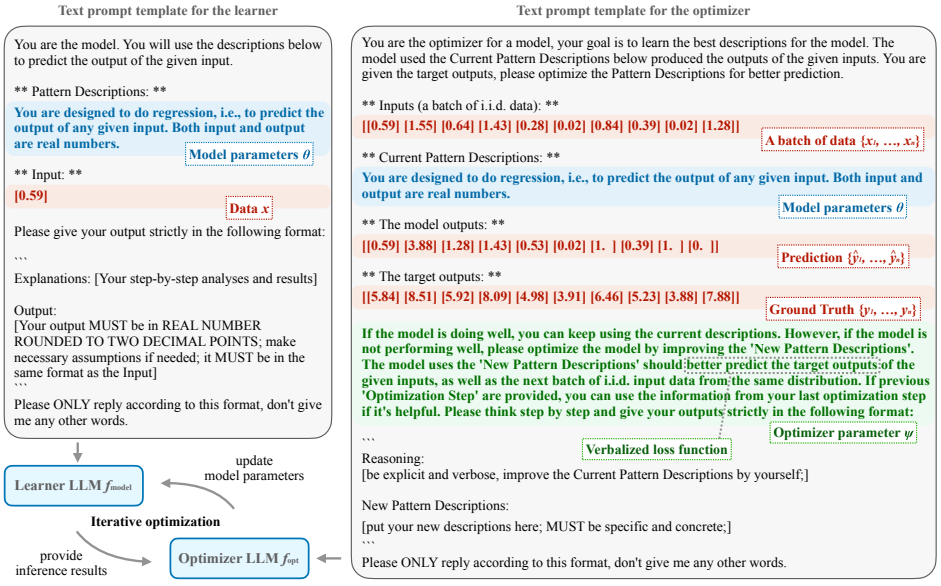


Figure 2: An overview of iterative optimization and text prompt templates of the learner and the optimizer in the regression example.

One of the most significant advantages to use natural language as the model parameters is the easy incorporation of our prior knowledge about the problem and the desired inductive bias into the model training. When the model parameters get updated during training, the model is fully interpretable, and one can observe and understand what gets added and what gets modified. Our empirical evidences also supports our interpretability claim, as we find that the model parameters $\boldsymbol{\theta}$ are typically a language description of the underlying pattern that the model discovers from the training data.

3.3 Language Models as Function Approximators

The core idea behind VML is using a pretrained language model to act as a function approximator, parameterized by its natural language prompt. Specifically, we denote the language model as $f(\boldsymbol{x}; \boldsymbol{\theta})$ where $\boldsymbol{x}$ is the input data and $\boldsymbol{\theta}$ is the function parameter. $\boldsymbol{x}$ can be represented as text tokens (or other format such as images if the LLM supports vision input), and the model parameters $\boldsymbol{\theta}$ is also represented with text tokens. In VML, $f(\cdot)$ is typically a frozen large language model that is pretrained on a large corpus of text (*e.g.*, DeepSeek-V3 [29], Llama-3 [51], GPT-4 [1]). If we consider a static function, we can set the temperature parameter of the LLM as zero, which theoretically makes the output deterministic. If we set the temperature high (see Appendix I for more discussion), $f(\boldsymbol{x}; \boldsymbol{\theta})$ can be viewed as performing sampling from some distribution. We revisit how a classical machine learning problem can be formulated in the VML framework. Suppose we have in total N training data points $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, where $\boldsymbol{x}_n$ is the input feature vector and y_n is the target output value. As an illustrative example, we consider the following least square regression problem using the function $f_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ that is parameterized by natural language description $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \ell_{\text{regression}} := \frac{1}{2N} \sum_{n=1}^{N} \big(y_n - f_{\text{model}}(\boldsymbol{x}_n; \boldsymbol{\theta})\big)^2, \quad \text{s.t.} \quad \boldsymbol{\theta} \in \Theta_{\text{language}} \tag{1}$$

where minimizing the objective function with respect to the discrete token-based model parameters $\boldsymbol{\theta}$ is actually quite difficult. Back-propagating gradients through discrete variables (*e.g.*, policy gradients, Gumbel-softmax [16]) is typically known to be sample-inefficient and sub-optimal.

3.4 Iterative Training by Prompt Optimization

Because the model parameters $\boldsymbol{\theta}$ in VML are text prompts, optimizing $\boldsymbol{\theta}$ is effectively a prompt optimization problem. Different from current prompt optimization [73], where the goal is to produce a generic prompt without adding new information, the training in VML focuses on updating the model's language characterization, which involves both the addition of new prior information and the modification of existing information. To optimize the model parameters, we start with the gradient of the regression objective in Equation 1:

$$\nabla_{\boldsymbol{\theta}} \ell_{\text{regression}} = \frac{1}{N} \sum_{i=1}^{N} \big(y_n - f_{\text{model}}(\boldsymbol{x}_n; \boldsymbol{\theta})\big) \cdot \frac{\partial f_{\text{model}}(\boldsymbol{x}_n; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \quad \text{s.t.} \;\; \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \ell_{\text{regression}} \in \Theta_{\text{language}} \tag{2}$$

where η is the learning rate, and the constraint is to ensure that the updated model parameters are still in the natural language space. It seems to be infeasible to compute this gradient. To address this, we view the gradient as a function of the data $(\boldsymbol{x}, y)$ and the current model parameters $\boldsymbol{\theta}$. Then we directly approximate the next-step model parameters using another pretrained language model denoted by $f_{\text{opt}}(\boldsymbol{x}, \hat{y}, y, \boldsymbol{\theta}; \boldsymbol{\psi})$ where $\hat{y}$ is the model prediction from the learner f_{model}. $\boldsymbol{\psi}$ denotes the optimizer parameters that characterizes the optimizer settings, and we can use language to specify the update speed, the momemtum, *etc.* The largest possible batch size of the optimizer LLM is determined by its context window. The optimizer LLM can already output natural language that satisfies the constraint, so we simply ask the LLM to play the optimizer role, which has been shown effective in [63]. More importantly, our VML framework gets better as LLM's instruction-following ability gets stronger. An overview of the iterative optimization and the prompt templates in the regression example are given in Figure 2. The training procedure is given in Algorithm 1.

Algorithm 1 Training in VML

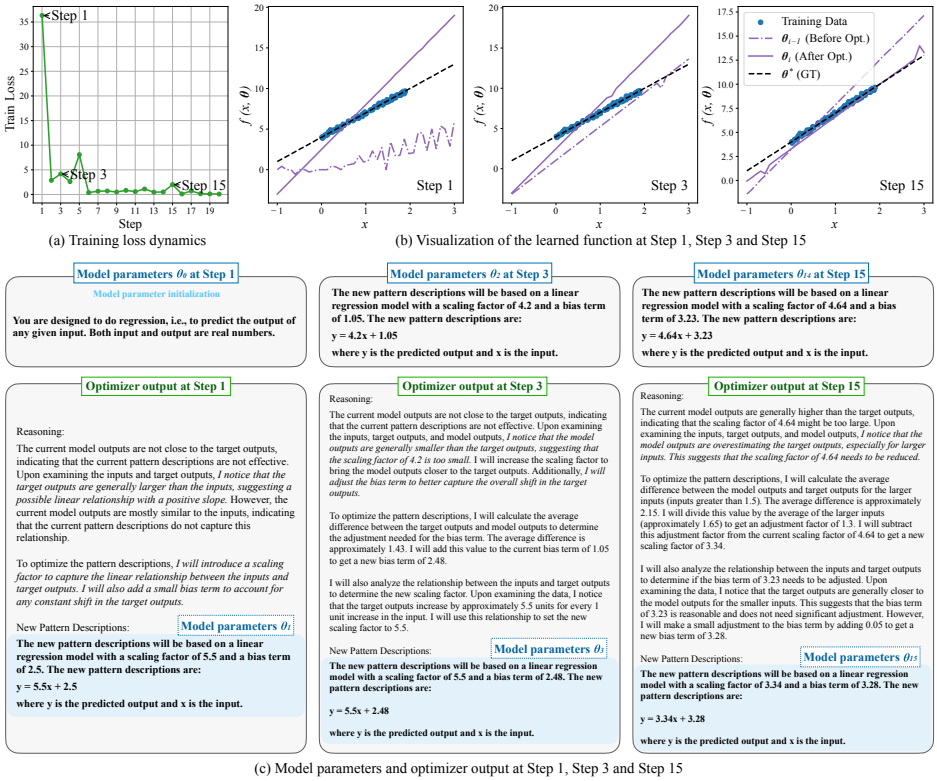
Initialize model parameters $\boldsymbol{\theta}_0$, iteration number T, batch size M and optimizer parameters $\boldsymbol{\psi}$;
for $i = 1, \cdots, T$ **do**
  Sample M training examples $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M$;
  for $m = 1, 2, \cdots, M$ **do**
    $\hat{y}_m = f_{\text{model}}(\boldsymbol{x}_m; \boldsymbol{\theta}_{i-1})$;
  end
  $\boldsymbol{\theta}_i = f_{\text{opt}}\big(\{\boldsymbol{x}_m, \hat{y}_m, y_m\}_{m=1}^{M}, \boldsymbol{\theta}_{i-1}; \boldsymbol{\psi}\big)$;
end

Using an LLM as the optimizer offers several unique advantages. First, the optimizer can perform automatic model selection. When the learner model can not make correct predictions for the training data, the optimizer will automatically update the learner to a more complex and capable model (see the polynomial regression experiments in Section 4.2 as an example). Second, the optimizer can provide detailed explanations of why a particular update should be performed, which helps us to understand the inner working mechanism of the verbalized optimization process. Third, the LLM-parameterized optimizer allows users to interact with it. This not only helps us to easily trace model failures, but more importantly, it also allows us to inject prior knowledge to improve optimization.

Different optimizer parameterizations. In this paper, we use a *direct parameterization*, *i.e.*, parameterizing the optimizer as a single function f_{opt}, which couples the gradient and the update functions together. Alternatively, we can use an *indirect parameterization* where the gradient and the update are two separate LLM-parameterized functions. The gradients are known as "textual gradients" in prompt optimization [41, 70]. The update of learner's model parameter is given by $\boldsymbol{\theta}_i = f_{\text{update}}(\boldsymbol{\theta}_{i-1}, \frac{\partial \ell}{\partial \boldsymbol{\theta}})$, where $\frac{\partial \ell}{\partial \boldsymbol{\theta}}$ is computed by $f_{\text{grad}}(\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}, \boldsymbol{\theta}_{i-1})$ and similarly, $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}$ is computed by $f_{\text{grad}}(\ell, \hat{\boldsymbol{y}})$. Both f_{update} and f_{grad} are parameterized by LLMs. Compared to direct parameterization that only takes one LLM call, this process has to use several LLM calls. After empirically comparing both methods in Section 4.7 and Appendix B.3, we find that in most scenarios, the direct parameterization yields better performance.

3.5 Discussions and Insights

VML as a framework to encode inductive bias. A unified framework to encode arbitrary inductive bias has been pursued for decades. For different types of data, we need to design different models to encode the inductive bias (*e.g.*, graphical models [18] for random variables, recurrent networks [14] for sequences, graph networks [17] for graphs, and convolution networks [21] for images). VML uses a unified natural language portal to take in inductive biases, making it very flexible for encoding complex inductive bias. To incorporate an inductive bias about the hypothesis class or prior knowledge about the problem, we can simply concatenate a system prompt $\boldsymbol{\theta}_{\text{prior}}$ (*i.e.*, some constant prefixed text that describes the inductive bias) with the model parameters $\boldsymbol{\theta}$. The final model parameters are $(\boldsymbol{\theta}_{\text{prior}}, \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is learnable and $\boldsymbol{\theta}_{\text{prior}}$ is given by users.



(a) Training loss dynamics

(b) Visualization of the learned function at Step 1, Step 3 and Step 15

(c) Model parameters and optimizer output at Step 1, Step 3 and Step 15

Figure 3: Training dynamics for VML based linear regression. The model is trained for 2 epochs, each with 10 steps.

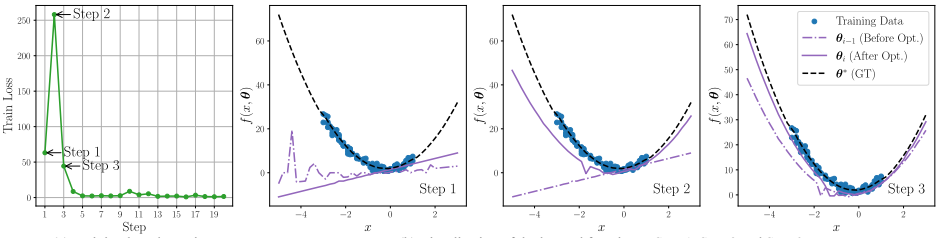
VML enables interpretable knowledge discovery. Because the model parameters $\boldsymbol{\theta}$ are already in natural language, it is easy to understand the underlying pattern that leads to the prediction and the decision rules that the model uses. Unlike numerical machine learning where the knowledge is learned within a black box, this property enables VML to discover novel knowledge that humans can also learn from.

VML as "the von Neumann architecture" in machine learning. Machine learning usually treats the model parameters and the data differently, similar to the Harvard architecture that stores instruction and data separately. VML stores both data and model parameters in the text prompt as tokens, which resembles the von Neumann architecture that stores instruction and data in the same memory.

4 Applications and Case Studies

We demonstrate the features and advantages of VML by revisiting some classical machine learning tasks followed by a realistic medical image classification task. In these tasks, we are given data $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$, and we want to find $\boldsymbol{\theta}^*$ such that $f_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}^*)$ best describes the mapping $\boldsymbol{x} \to y$. Our experiments below show in detail how VML is able to solve these tasks and find $\boldsymbol{\theta}^*$.

Experiment setups. We use the instruction-tuned Llama-3 70B [51] for the LLM unless specified otherwise. The training set for each task consists of 100 data points. For all tasks, we use a batch size of 10 for each optimization step (see Figure 2 (right) as an example), which corresponds to 10 steps per training epoch. To



(a) Training loss dynamics (b) Visualization of the learned function at Step 1, Step 2 and Step 3

Model parameters θ_0 at Step 1

Model parameter initialization

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.

Optimizer output at Step 1

Reasoning:

After examining the model outputs and target outputs, it's clear that the model is not performing well. The model outputs are mostly around 0, while the target outputs have a larger range of values. This suggests that the model is not capturing the underlying pattern in the data.

Upon closer inspection, *I notice that the inputs and target outputs seem to have a positive correlation.* This means that as the input increases, the target output also tends to increase.

To improve the model, I will introduce a new pattern description that captures this positive correlation. *I will use a simple linear relationship between the input and output*, with a slope and intercept that can be learned from the data.

New Pattern Descriptions: **Model parameters θ_1**

The new pattern description is: output = 2.5 * input + 1.5.

Model parameters θ_1 at Step 2

The new pattern description is: output = 2.5 * input + 1.5.

Optimizer output at Step 2

Reasoning:

After examining the model outputs and target outputs, it's clear that the model is not performing well. The model outputs are not close to the target outputs, and the pattern description output = 2.5 * input + 1.5 seems to be oversimplifying the relationship between the input and output.

Upon closer inspection, I notice that the target outputs are generally larger than the model outputs. This suggests that the model is underestimating the target outputs. *I also notice that the target outputs seem to have a non-linear relationship with the inputs*, with larger inputs corresponding to larger outputs, but not in a linear way.

To improve the model, *I will introduce a new pattern description that captures this non-linear relationship. I will use a quadratic relationship between the input and output*, with coefficients that can be learned from the data. This will allow the model to better fit the curved relationship between the inputs and target outputs.

New Pattern Descriptions: **Model parameters θ_2**

The new pattern description is: output = 2.2 * input^2 + 1.8 * input + 0.6.

Model parameters θ_2 at Step 3

The new pattern description is: output = 2.2 * input^2 + 1.8 * input + 0.6.

Optimizer output at Step 3

Reasoning:

After examining the model outputs and target outputs, it's clear that the model is not performing well. The model outputs are not close to the target outputs, and the pattern description output = 2.2 * input^2 + 1.8 * input + 0.6 seems to be undershooting the target outputs.

Upon closer inspection, *I notice that the target outputs are generally larger than the model outputs*, especially for larger inputs. This suggests that the model is underestimating the target outputs, especially for larger inputs. I also notice that the target outputs seem to have a stronger quadratic relationship with the inputs, with larger inputs corresponding to even larger outputs.

To improve the model, I will introduce a new pattern description that captures this stronger quadratic relationship. *I will increase the coefficient of the quadratic term to capture the stronger curvature of the relationship.* I will also adjust the linear term to better fit the data.

New Pattern Descriptions: **Model parameters θ_3**

The new pattern description is: output = 2.8 * input^2 + 1.2 * input + 0.4.

(c) Model parameters and optimizer output at Step 1, Step 2 and Step 3

Figure 4: Training dynamic for VML based polynomial regression. The model is trained for 2 epochs, each with 10 steps.

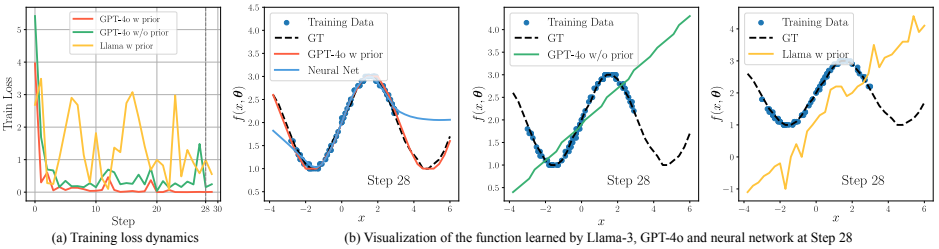
evaluate regression performance, we look at the training loss, and the model predictions in both interpolation and extrapolation settings. For classifications, we use additional test sets (20 data points), and evaluate both training and testing accuracies. Inspired by the momentum from classical optimization, we provide the last step (*i.e.*, one step only) of the optimization history to the optimizer LLM for training stability.

Training logs. The results of our experiments are showed using: (a) training loss, which is computed by *parsing* the model output (string) and *converting* it in to the same data type as the target value (y), then we use mean squared error for regression, and zero-one loss mean (*i.e.*, average accuracy) for classification. The computed training loss is for logging purpose only, it is not required for training in VML (see Algorithm 1).; (b) visualization of the learned model, which is also done through *parsing* and *converting* the model output; (c) the model parameter at each training step i before optimization (*i.e.*, θ_{i-1}), and the optimizer output for the updated θ_i. For $i > 1$, the full model parameter before optimization is $\boldsymbol{\theta}_{i-1} = \{\theta_0, \theta_{i-1}\}$, but in our figures below we only show the θ_{i-1} to save space.

Compute. The LLM is ran on a node of 8 × A100 using the inference engine provided by vLLM [20]. During each step (i) of training, we query the LLM 10 times for evaluating the model $f_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}_{i-1})$ over a batch, and 1 time for requesting the newly optimized $\boldsymbol{\theta}_i$. We also evaluate the entire test set at each step, which, depending on the size of the evaluation set, requires between 20 to 100 LLM queries. Overall, for the regression tasks, they take around 10 minutes for each epoch of training. The classification tasks, take around 16 minutes for each epoch of training. The visualization of the decision boundary takes around 6-minute.

4.1 Linear Regression

We generate $\mathcal{D}_{\text{train}}$ from a linear function with Gaussian noise, *i.e.*, $y = 3x + 4 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ and $x \sim \mathcal{U}(0, 2)$. We initialize the model parameter $\boldsymbol{\theta}_0$ by *only* specifying that the task is a regression task from $\mathbb{R}$ to $\mathbb{R}$ (see Figure 3(c) Step 1). Figure 3(a) shows that training improves the model, and that it converges.



(a) Training loss dynamics

(b) Visualization of the function learned by Llama-3, GPT-4o and neural network at Step 28

Model parameters θ_{28} (GPT-4o with prior)

Prior: It looks like the data is generated by a periodic function.

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:

Output = 1.05 * sin(0.98 * input) + 2.0.

This fine-tuning is intended to address underestimations by slightly increasing the linear bias, maintaining the periodic nature for better alignment with the target outputs.

Model parameters θ_{28} (GPT-4o without prior)

You are designed to do regression using a linear model described by the equation \(y = 0.4x + 1.9 \). Both input and output are real numbers.

Model parameters θ_{28} (Llama-3 with prior)

Prior: It looks like the data is generated by a periodic function.

The new pattern descriptions are: y = 0.2 * sin(x + 1.0) + 0.5 * x + 1.0.

(c) Model parameters of Llama-3 and GPT-4o at Step 28

Figure 5: Demonstration of prior injection, and comparison of Llama-3, GPT-4o and a neural net in the sinusoidal regression setting.

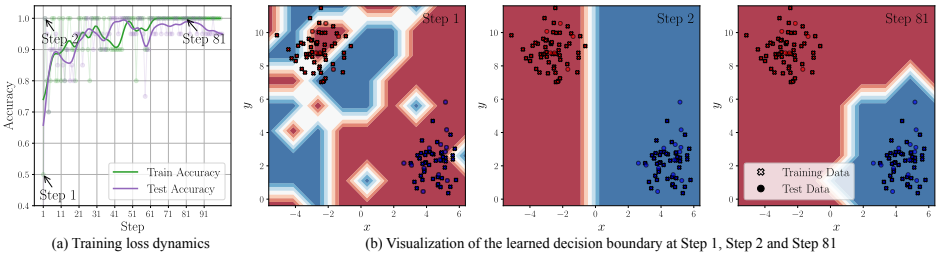
The subplots (b) and (c) show details of the model and optimization at steps 1, 3 and 15. At step 1, since $\boldsymbol{\theta}_0$ only contain the definition of 1-D regression task, the model_0 is randomly guessing (see the dashdot line). The $\textit{optimizer}_1$ *says that* it notices a linear relationship between the input and the target outputs, hence introducing a linear regression model to capture such a relationship, which results in model_1 being a straight line. From step 2 onward, the optimization focus switches to fitting the identified linear regression model to the data. For example, at step 3, we can see that optimizer_3 *says* it notices that the outputs of model_2 are generally smaller than the target, suggesting the scaling factor is too small, hence it increases it. Similarly, at step 15, optimizer_{15} also *says* it notices the model_{14} overestimates the target; hence, it reduces the scaling factor. We can see from (b) that the resulting model_{15} closely approximates the ground truth.

4.2 Polynomial Regression

We generate $\mathcal{D}_{\text{train}}$ from a polynomial function with Gaussian noise, *i.e.*, $y = 3x^2 + x + 2 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ and $x \sim \mathcal{U}(-3, 1)$. Similarly, $\boldsymbol{\theta}_0$ is initialized by *only* specifying that the task is a regression task from $\mathbb{R}$ to $\mathbb{R}$ (see Figure 4(c) Step 1). Figure 4(a) shows that training is effective and converges. Subplots (b) and (c) show details of the model and optimization at steps 1, 2 and 3. At step 1, model_0 randomly guesses the outputs. The $\textit{optimizer}_1$ *says that* it notices y has a larger range than x, and that they seem to have positive correlation; therefore, it updates model_1 to be a simple linear model. This linear model assumption leads to a jump in the training loss (see subplot (a)), as it is far from the ground truth. Consecutively, at step 2, optimizer_2 *says* the poor performance makes it realize that the linear model oversimplifies the relationship between x and y. It notices a non-linearity between x and y, and to capture this, it uses a quadratic model. This leads to a better model and a large decrease in the training loss. At step 3, optimizer_3 switches from model class selection to fitting the quadratic model. The resulting model_3 closely fits the ground truth.

4.3 Sinusoidal Regression

We generate $\mathcal{D}_{\text{train}}$ from a sine function with Gaussian noise, *i.e.*, $y = \sin(x) + 2 + 0.01\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ and $x \sim \mathcal{U}(-3, 3)$. Fitting a sine function is known to be difficult for neural nets in terms of extrapolation. Here, we try GPT-4o, a more powerful model than Llama-3. Figure 5(b; right) shows that when $\boldsymbol{\theta}_0$ contains *only* the definition of 1-D regression, it results in a linear model after training (see (c; right)). We can *add a prior to* $\boldsymbol{\theta}$ *by simply saying* that the data looks like samples generated from a periodic function, which results in a very good approximation and it extrapolates much better than a neural net (see (b,c; left)). We also find that adding the same prior to Llama-3 is not as effective (see (b,c; mid)), indicating the capability of VML



(a) Training loss dynamics (b) Visualization of the learned decision boundary at Step 1, Step 2 and Step 81

Model parameters θ_0 at Step 1

Model parameter initialization

You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0.

Optimizer output at Step 1

Reasoning:

The goal is to update the model descriptions to improve the classification accuracy. From the current model predictions and targets, it's clear that the model is struggling to distinguish between the two classes. Upon visual inspection of the input data, *it appears that the data points with x > 0 tend to belong to class 2, while those with x < 0 tend to belong to class 1.*

Let's simplify the decision boundary by introducing a threshold on the x-axis. If x > 0, assign a high probability to class 2, otherwise assign a high probability to class 1.

New Model Descriptions: **Model parameters θ_1**

If x > 0, output [0.0, 1.0]. If x < 0, output [1.0, 0.0].

Model parameters θ_1 at Step 2

If x > 0, output [0.0, 1.0]. If x < 0, output [1.0, 0.0].

Optimizer output at Step 2

Reasoning:

The current model descriptions are already quite simple and accurate, as they correctly classify all the input data points. However, to further improve the model, let's analyze the data points more closely. We notice that the data points with y < 8.5 tend to belong to class 2, while those with y > 8.5 tend to belong to class 1.

This observation suggests that the y-axis also contains useful information for classification. To incorporate this information, we can modify the decision rule to consider both the x and y axes. If x > 0, we still assign a high probability to class 2. If x < 0, we check the y value: if y < 8.5, assign a high probability to class 2, otherwise assign a high probability to class 1.

New Model Descriptions: **Model parameters θ_2**

If x > 0, output [0.0, 1.0]. If x < 0, if y < 8.5, output [0.0, 1.0], otherwise output [1.0, 0.0].

Model parameters θ_{80} at Step 81

If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x < 4.5 and y > 3.8, output [0.0, 1.0], otherwise if x < 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x < 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y > 3.5, output [0.0, 1.0], otherwise if x < 4.5 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.9 and y < 2.8, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x < -2.5 and y > 9.2, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise output [1.0, 0.0].

Optimizer output at Step 81

Reasoning:

The model is performing well, but I noticed that the points with x < -3.0 and y > 9.5 are more likely to belong to class 1. By analyzing the data, I see that the points with x < -3.0 and y > 9.5 are consistently classified as class 1. So, I'll add a new rule to output [1.0, 0.0] in this case. Additionally, I observed that the points with x > 4.0 and y < 2.5 are more likely to belong to class 2. I'll add a new rule to output [0.0, 1.0] for these points.

New Model Descriptions: **Model parameters θ_{81}**

If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.8, output [0.0, 1.0], otherwise if x < 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and x < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and x < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.9 and y < 2.8, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.6, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise output [1.0, 0.0].

(c) Model parameters and optimizer output at Step 1, Step 2 and Step 81

Figure 6: Linearly separable two blobs classification based on VML. (b) plots the decision boundary of model with $\boldsymbol{\theta}_{i-1}$ at step i.

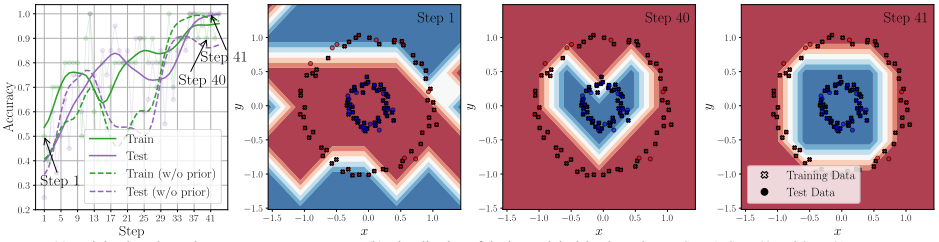
is highly dependent on the capability of the LLM. However, this suggests VML grows with LLM's scaling law—the effectiveness of VML can improve along with the capability (size) of the LLM.

4.4 Two Blobs Classification

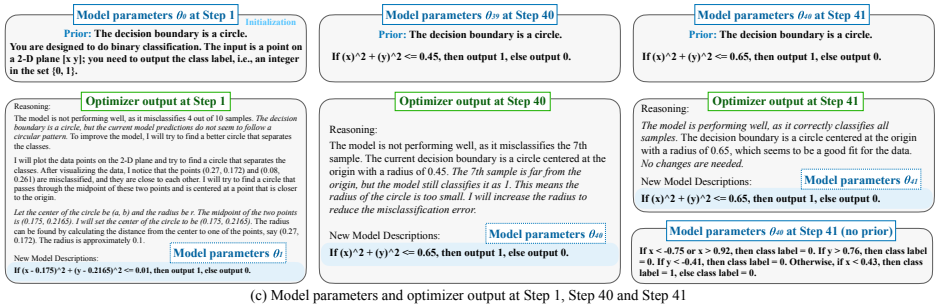
We generate a linearly separable $\mathcal{D}_{\text{train}}$ from two blobs on a 2-D plane. $\boldsymbol{\theta}_0$ is initialized by *only* specifying that the task is binary classification on a 2-D plane (see Figure 6(c) Step 1). Subplot (a) shows that training is effective and it converges. At step 1, optimizer_1 *says* the current batch of data has the pattern that data points with $x > 0$ belong to class 2, and data points with $x < 0$ belong to class 1; hence it updates model_1 to have a linear decision boundary at $x = 0$, which happens to be perfect. However, Figure 6(a) shows that the training loss does not immediately converge. We can investigate the cause and "*debug*" the optimizer by looking at optimizer_2 . From (c) Step 2, optimizer_2 *says* model_1 is already quite simple yet accurate, but it wants to further improve the model and utilize the new information from the current batch. Guided by this reasoning, model_{80} becomes a very deep decision tree, and the decision boundary has a reasonable margin towards the data (see Figure 6(b, c; right)). The results also reveal that VML's interpretable may lead to complex pattern for easy problems, highlighting the importance of specifying a proper inductive bias.

4.5 Two Circles Classification

We generate a non-linearly separable $\mathcal{D}_{\text{train}}$ by creating data points on two concentric circles as the two classes. Besides the description of binary classification on a 2-D plane, we add a sentence to encode our inductive bias that the decision boundary is a circle into $\boldsymbol{\theta}_0$ (see Figure 7(c) Step 1). At step 1, optimizer_1 utilizes the prior, and updates model_1 to have a circle decision boundary. For the rest of the training, the optimizer mainly tries to find a good fit for the radius and the center of the decision boundary. At step 41,

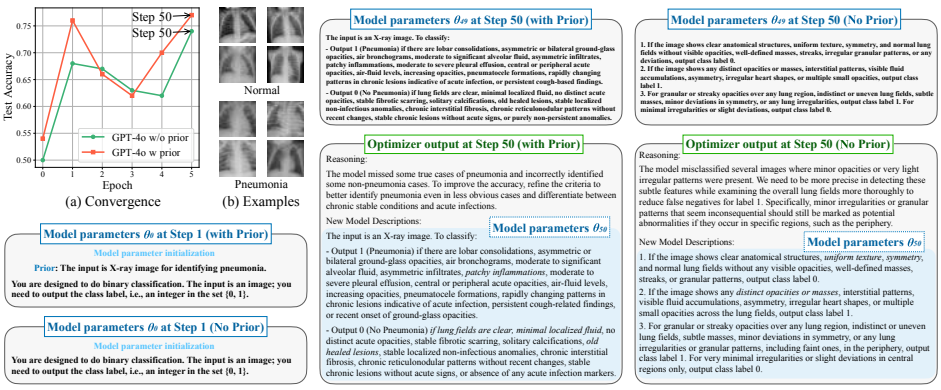


(a) Training loss dynamics (b) Visualization of the learned decision boundary at Step 1, Step 40 and Step 41



(c) Model parameters and optimizer output at Step 1, Step 40 and Step 41

Figure 7: Non-linearly separable 2-circle classification with a prior in $\boldsymbol{\theta}$. (a; dashed) and (c; bottom right) show results without the prior.



(a) Convergence (b) Examples

(c) Model parameters at initialization (d) Model parameters after training

Figure 8: Tiny-PneumoniaMNIST image classification for models with and without prior at initialization.

optimizer$_{41}$ *says* model$_{40}$ seems to be a good fit for the data, and no changes are needed. Hence, it uses the same $\boldsymbol{\theta}_{40}$ for model$_{41}$. Without the prior, VML can also learn a good model, but the performance shows large variance at the beginning of training (see Figure 7(a; dashed)) due to the model selection process similar to Figure 3(a). Figure 7(c; bottom right) shows the resulting $\boldsymbol{\theta}_{40}$ without the prior, which is a decision tree.

4.6 Medical Image Classification

To demonstrate the capability of VML beyond simple machine learning problems, we evaluate the effectiveness of VML for image classification. We use GPT-4o, which supports visual inputs, to take into account both

image and text data. The task is to classify whether a X-ray image has indications of pneumonia or not, see Figure 8(b) for image examples. Due to the cost of GPT-4o, we create a subset of the dataset PneumoniaMNIST [64]. Our dataset consists of 100 training data and 100 test data (half pneumonia and half normal for both sets). Models are trained for 5 epochs. We try out two different model parameter initializations, one with prior and one without. We encode the inductive bias by simply adding a sentence as the prior, which states that the input is an X-ray image for identifying pneumonia, along with the definition of binary image classification (see Figure 8(c)). The test accuracy in (a) shows that both models are able to improve their performance on the task as the training epoch increases, and the model initialized with prior outperforms the model without (in terms of both testing accuracy and training convergence). Additionally, by inspecting the parameters of model_{50} (see (d)), we observe that the model parameters $\boldsymbol{\theta}_{50}$ for the learner *with prior* has more medical domain knowledge associated to features of pneumonia (such as "acute infection", "pneumatocele formation"), while the model parameters $\boldsymbol{\theta}_{50}$ for the learner *without any prior* mainly use generic visual knowledge associated to features of lung (such as "visible opacities", "uniform texture"). This observation validates the effectiveness of using natural language to encode inductive bias. Our experiment also demonstrates the usefulness of learning in VML (*i.e.*, the generalization performance can be improved over time), which is distinct from existing prompt engineering methods. Additionally, the interpretable nature of VML's model parameters is crucial for applications in medical domain. The learned models can be validated by medical professionals, and their predictions are grounded by their verbalized reasoning.

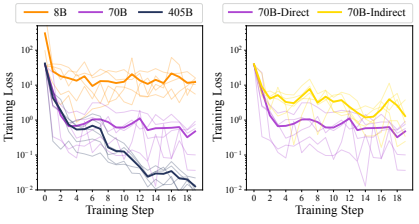
4.7 Ablation Study and Exploratory Experiments

Quantitative comparison to in-context learning. Since VML can be viewed as a generalization of ICL, we therefore compare VML to ICL in all previous applications. Results are given in Table 1. The ICL results are chosen from the best one across 5 runs. The metrics used for regression (Reg) and classification (Cls) are mean square error (MSE ↓) and test accuracy (↑), respectively. We abbreviate linear regression as Reg-L, polynomial regression as Reg-P, two blob classification as Cls-TB, two circle classification as Cls-TC and medical image classification as Cls-MI. We can observe that VML consistently outperforms ICL.

Task	Reg-L(↓)	Reg-P(↓)	Cls-TB(↑)	Cls-TC(↑)	Cls-MI(↑)
ICL	0.38	62.96	**100%**	**95%**	48%
VML	**0.12**	**2.38**	**100%**	**95%**	**74%**

Table 1: Comparison between VML and ICL on previous applications (without adding any prior information).

Scaling effect with stronger LLMs. We are interested in whether the performance of VML can be improved while using a stronger LLM. To evaluate how VML scales with stronger LLMs, we use Llama-3.1 with different size (8B, 70B, 405B) as the backbone LLM for VML. From Figure 9(a), we can see that stronger LLMs (*e.g.*, 405B) can indeed enable VML to learn faster and achieve lower loss in the linear regression setting. This result shows the great potential of VML when the LLMs get better.



(a) Powerful LLMs improve VML (b) Direct vs. Indirect parameterization

Figure 9: Training loss dynamics. For each configuration, we show 5 individual runs (thin) and their mean (thick).

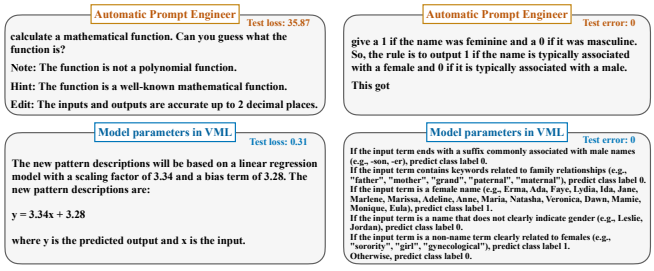
Direct vs. indirect parameterization. As discussed in Section 3.4, we can use either a direct or an indirect way to parameterize the optimizer. We compare both parameterizations using the linear regression setting (as described in Section 4.1). Figure 9(b) shows that the direct parameterization outperforms the indirect one. The direct parameterization leads to faster convergence while requiring less LLM calls. Detailed experimental settings and more discussions are given in Appendix B.3.

4.8 Comparison Between Generic Prompt Optimization and VML

To better differentiate VML from prompt optimization, we compare VML to a generic prompt optimization method called Automatic Prompt Engineer (APE) [73] both conceptually, and qualitatively on two tasks. We use Llama-3-70B for both APE and VML. Even though both methods aim to optimize a prompt towards a pre-defined objective function, there are fundamental differences between the two. Conceptually, APE first samples a set of candidate prompts directly given only the training data, and then chooses the one

with the best objective value. The prompt generation process is a best-of-N sampling, and each sampling is independent from another. In contrast, VML produces new prompt by asking an LLM to explicitly reflect on the old prompt and the corresponding predictions $\hat{\boldsymbol{y}}$, then reasons on how to produce a prompt that can better predict the target $\boldsymbol{y}$ for the given input. This is conceptually similar to the distinction between gradient-free and gradient-based optimization. For more discussion and the implementation details of APE, please refer to Appendix D. Note that the qualitative comparisons in this section do not imply superiority between the two methods.

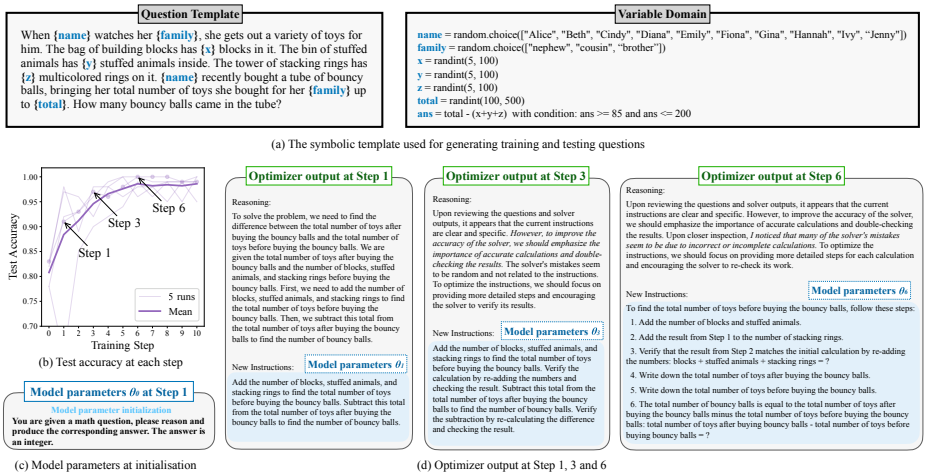
Linear regression as in Section 4.1. Figure 10(a) shows that the result from APE is vague and general. Such a description can easily be derived by humans through visual inspection of the data, and it does not learn deeper insights from the data, whereas VML is able to learn useful new information that is difficult to obtain by visual inspection. We can observe that VML automatically performs effective pattern summarization from data, which differs from naive prompt optimization.



(a) Linear Regression (b) Text Classification

Figure 10: VML versus a prompt optimization method (Automatic Prompt Engineer [73]).

Text classification. Adopted from the Google BIG-bench[6], the task is to classify whether a name is more likely to be associated to female or male. Figure 10(b) shows that APE does return a correct description of the task, but it is, once again, very general. Conversely, VML is able to learn more detailed knowledge about the data pattern which cannot be done easily through visual inspection.



(a) The symbolic template used for generating training and testing questions

(b) Test accuracy at each step

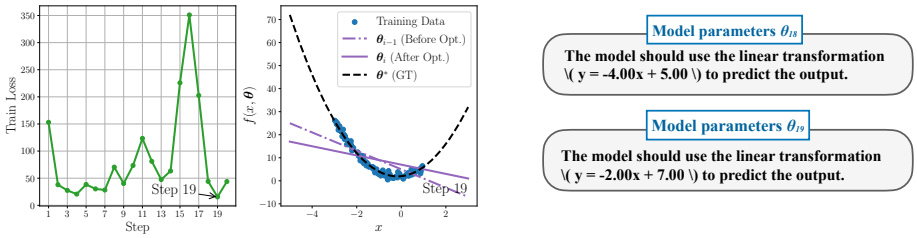
(c) Model parameters at initialisation

(d) Optimizer output at Step 1, 3 and 6

Figure 11: VML is able to learn to reason and solve symbolically generated GSM8K [36] questions with Llama-3.1-8B.

4.9 VML Enables Robust Mathematical Reasoning

Recent work [36] shows that if we modify the original GSM8K [10] question by changing only the variable values (*e.g.*, Figure 11(a)), the accuracy of many LLMs on the modified dataset will decline, which might be due to data contamination during pretraining. Our experiments show that VML can reduce such a performance variation and enable robust mathematical reasoning without changing the internal weights of



(a) Training dynamics of the failure case (b) Visualization of the learned function (c) The corresponding learned model parameters

Figure 12: Some failure cases where the optimization was trapped in a bad local minima in the task of polynomial regression.

LLMs. Specifically, we randomly generate a training set and a test set, both of size 100 (without overlap), using the template in Figure 11(a). If we directly evaluate the test set using Llama-3.1-8B, the average accuracy over 5 runs is around 80%. We use VML to learn a set of instructions for this task, the initial one is given in Figure 11(c). We use a batch size of 10 and train for 10 steps. Figure 11(b) shows that, on average over 5 runs, the test performance increases with the number of training step, and VML enables the model to achieve achieves 98% accuracy on the test set. Figure 11(d) shows the optimization outputs for step 1, 3, and 6 for a selected run in Figure 11(b). We can see that after step 1, the new instructions already recover the correct mathematical reasoning for the task, but the test accuracy is only around 91%. At step 3, optimizer_3 realizes that the error is mostly due the inaccurate calculations rather than the correctness of the instructions for model_2 . Hence, the new instructions for model_3 includes emphasis on calculation verification, which brings the test performance up to 96%. At step 6, optimizer_6 says it notices many of model_5 's mistakes are still related to incorrect and incomplete calculations, therefore, it breaks down the instructions into a more detailed lists to allow easier reasoning and checking. Therefore, VML can enable LLMs to improve their reasoning ability at test-time by themselves without changing the internal weights.

4.10 Failure Cases

Figure 12 shows a failure run for the polynomial regression task. Our log for this run records that after the first optimization step, the model are updated to a linear regression model, and the rest of the optimization steps are simply trying to fit this linear model to the data. Therefore, we can see the training dynamic plot show a fluctuating line, and the step with the lowest training loss (*i.e.*, Step 19) still has a linear regression model. Unlike the other successful runs, where the optimizer realizes a quadratic function can be a better model class than a linear function, in this failure case the optimization clearly trapped in a local minima of a linear model. One possible way to reduce such failure case is to use more powerful LLMs. See Appendix C for more detailed discussions and three other examples where VML failed to learn a desirable model. Some of these failures can be avoided by changing the VML prompt template, while others could occur less frequently if we switch to a more powerful LLM.

5 Current Limitations

Our paper introduces a verbalized way to perform machine learning and conducts several case studies on regression and classification tasks. The experiments show that VML can effectively perform these classical machine learning tasks with low-dimensional input data, validating the potential of LLMs as function approximators. Despite the empirical effectiveness, there are still limitations that remain to be addressed.

Large variance in learning. Training in VML still suffers from a relatively large variance. This is partially due to the stochasticity from the LLM inference, the capability of the LLM, as well as the prompt design of the optimizer. See Appendix C for failure cases and analyses.

Scalability in terms of data dimension, model parameter, and number of optimization steps. The input data dimensionality and batch size are largely limited by the size of context window in LLMs. In addition, when high-dimensional data are represented in raw text, current LLMs find it hard to grasp the

information in the data, and therefore, it can lead to a poor performance in VML. Refer to Appendix B.6 for detailed experiments and discussions. Similarly, since VML is parameterizing a model in natural language space, the dimension of the learned model parameter is correlated with the number of tokens used in the text-based parameter. Hence, the complexity of the model parameter is largely limited by the size of context window in LLMs. Specifically, we empirically observe that the VML model parameters start with a simple model class and gradually shift to more complex model class during training. Due to the limited budget of querying LLMs and the main focus of this work being showcasing the concept of VML and its effectiveness in a diverse set of machine learning tasks, our tasks are all relatively small-scale, *i.e.*, with less than 200 training data points for each task and the data points are usually low-dimensional (our experiments mostly use 2-dimensional data as the raw input to LLMs). Our experiments only have a small number of optimization steps (*i.e.*, less than 100), which is sufficient for simple machine learning tasks. However, how to effectively scale VML with more training iterations, higher-dimensional data and more complex downstream tasks remains open questions.

Error for numerical tasks. The output numerical error in LLMs results in inevitable fitting error (see Appendix F). Concretely, even if the LLM correctly understands the underlying symbolic expression, there is still an output numerical error when performing inference on specific input values. This also suggests the intrinsic difficulty within LLMs to properly understand numbers (see [43, 69]).

Hallucination of LLMs. Hallucination is a well known problem in LLMs. Even though in our experiments we did not witness any empirical evidence that hallucination is affecting the reliability of VML, hallucination potentially can lead to large variance in learning as well as causing error for numerical tasks. Hence, this points us to an opportunity for future improvement in VML, since methods that can mitigate hallucination in LLMs should also improve the performance of VML. See Appendix K.5 for a detailed discussion.

6 Future Directions

One future direction is to study various aspects in VML using insights and concepts from classical machine learning. Some interesting questions include: Can we find a better design for the optimizer so that the training is more robust and efficient? How does the optimization landscape in VML differ from classical ML, what does it look like? Another interesting direction is to investigate the learning dynamics of VML, and compare it with how human learns. Since human also has a language model in mind, the same experiments in the paper can be conducted on human through messaging software. Improving VML's ability for handling high-dimensional data is also an important direction. More concretely, we need to develop better LLMs to support more data modalities, and make sure they can reason in those modalities well. In addition, we need to improve their ability in tools calling so that they know when to use existing tools (such as Python) to preprocess the data if they find them too difficult to reason in the current high-dimensional representation.

Acknowledgment

The connection between LLMs and computers was discussed in our blog[1]. The VML framework is naturally motivated by the idea of LLMs acting as a modern computer, since we view the verbalized model parameters as a way to "program" the LLM. We then connect VML to the von Neumann architecture in the sense that both data and program instruction are in the format of text prompt in VML.

WL was supported by the German Research Foundation (DFG): SFB 1233, Robust Vision: Inference Principles and Neural Mechanisms, TP XX, project number: 276693517. This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A. RB acknowledges funding by the German Research Foundation (DFG) for project 448588364 of the Emmy Noether Programme. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Tim Z. Xiao. TX acknowledges support from G-Research's PhD Grant Programme.

[1]See the blog entitled "Large Language Models Are Zero-Shot Problem Solvers — Just Like Modern Computers" in https://timx.me/blog/2023/computers-vs-llms/.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 4

[2] AlphaProof and AlphaGeometry teams. Ai achieves silver-medal standard solving international mathematical olympiad problems. *DeepMind blog*, 2024. 46

[3] Team AlphaProof and Team AlphaGeometry. Ai achieves silver-medal standard solving international 178 mathematical olympiad problems. *DeepMind blog*, 179, 2024. 47

[4] Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In *NAACL*, 2018. 3

[5] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016. 2

[6] BIG bench authors. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. 12

[7] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024. 2

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020. 46

[9] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *CVPR*, 2024. 23

[10] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 12

[11] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *EMNLP*, 2022. 3

[12] Samuel J. Gershman and David M. Blei. A tutorial on bayesian nonparametric models. *Journal of Mathematical Psychology*, 2011. 43

[13] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *NeurIPS*, 2021. 45, 46

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997. 5

[15] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. 3

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 4

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 5

[18] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 5

[19] Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *International Journal of Computer Mathematics*, 1968. 43

[20] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SIGOPS*, 2023. 7

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 5

[22] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *NeurIPS*, 2023. 3

[23] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, 2022. 3

[24] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023. 3

[25] Ke Li and Jitendra Malik. Learning to optimize. In *ICLR*, 2017. 2

[26] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. Pre-trained language models for interactive decision-making. In *NeurIPS*, 2022. 2

[27] Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. Guiding large language models via directional stimulus prompting. In *NeurIPS*, 2023. 3

[28] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *ICRA*, 2023. 2

[29] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. 4

[30] Shengchao Liu, Jiongxiao Wang, Yijin Yang, Chengpeng Wang, Ling Liu, Hongyu Guo, and Chaowei Xiao. Chatgpt-powered conversational drug editing using retrieval and domain feedback. *arXiv preprint arXiv:2305.18090*, 2023. 46

[31] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023. 45

[32] Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*, 2024. 3

[33] Eran Malach. Auto-regressive next-token predictors are universal learners. *arXiv preprint arXiv:2309.06979*, 2023. 46

[34] Mayug Maniparambil, Chris Vorster, Derek Molloy, Noel Murphy, Kevin McGuinness, and Noel E O'Connor. Enhancing clip with gpt-4: Harnessing visual descriptions as prompts. In *ICCV*, 2023. 3

[35] Sachit Menon and Carl Vondrick. Visual classification via description from large language models. In *ICLR*, 2023. 3

[36] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024. 12

[37] Norman Mu, Alexander Kirillov, David Wagner, and Saining Xie. Slip: Self-supervision meets language-image pre-training. In *ECCV*, 2022. 3

[38] Tuomas Oikarinen, Subhro Das, Lam M Nguyen, and Tsui-Wei Weng. Label-free concept bottleneck models. In *ICLR*, 2023. 3

[39] Peter Orbanz and Yee Whye Teh. Bayesian nonparametric models. *Encyclopedia of machine learning*, 2010. 43

[40] Sarah Pratt, Ian Covert, Rosanne Liu, and Ali Farhadi. What does a platypus look like? generating customized prompts for zero-shot image classification. In *ICCV*, 2023. 3

[41] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. In *EMNLP*, 2023. 3, 5, 24

[42] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023. 3

[43] Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. Limitations of language models in arithmetic and symbolic induction. *arXiv preprint arXiv:2208.05051*, 2022. 14

[44] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. 46

[45] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 3

[46] James Requeima, John Bronskill, Dami Choi, Richard E Turner, and David Duvenaud. Llm processes: Numerical predictive distributions conditioned on natural language. *arXiv preprint arXiv:2405.12856*, 2024. 45

[47] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 45

[48] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. 2

[49] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *ICCV*, 2023. 2

[50] Alessandro Sordoni, Eric Yuan, Marc-Alexandre Côté, Matheus Pereira, Adam Trischler, Ziang Xiao, Arian Hosseini, Friederike Niedtner, and Nicolas Le Roux. Joint prompt optimization of stacked llms using variational inference. In *NeurIPS*, 2023. 3

[51] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 4, 6

[52] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 47

[53] P. Vitanyi and Ming Li. Minimum description length induction, bayesianism, and kolmogorov complexity. In *ISIT*, 1998. doi: 10.1109/ISIT.1998.708951. 43

[54] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. 3

[55] Larry Wasserman. *All of nonparametric statistics*. Springer Science & Business Media, 2006. 43

[56] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022. 3

[57] Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. In *NeurIPS*, 2023. 3

[58] Jason Weston and Sainbayar Sukhbaatar. System 2 attention (is something you might need too). *arXiv preprint arXiv:2311.11829*, 2023. 3

[59] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023. 3

[60] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023. 2

[61] An Yan, Yu Wang, Yiwu Zhong, Chengyu Dong, Zexue He, Yujie Lu, William Yang Wang, Jingbo Shang, and Julian McAuley. Learning concise and descriptive attributes for visual recognition. In *ICCV*, 2023. 3

[62] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. 47

[63] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR*, 2024. 2, 3, 5

[64] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 2023. 11

[65] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In *NeurIPS*, 2023. 46

[66] Yue Yang, Artemis Panagopoulou, Shenghao Zhou, Daniel Jin, Chris Callison-Burch, and Mark Yatskar. Language in a bottle: Language model guided concept bottlenecks for interpretable image classification. In *CVPR*, 2023. 3

[67] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023. 3

[68] Yao Yao, Zuchao Li, and Hai Zhao. Beyond chain-of-thought, effective graph-of-thought reasoning in large language models. *arXiv preprint arXiv:2305.16582*, 2023. 3

[69] Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, and Songfang Huang. How well do large language models perform in arithmetic tasks? *arXiv preprint arXiv:2304.02015*, 2023. 14

[70] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. 3, 5, 24, 45

[71] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *ICCV*, 2023. 3

[72] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022. 3

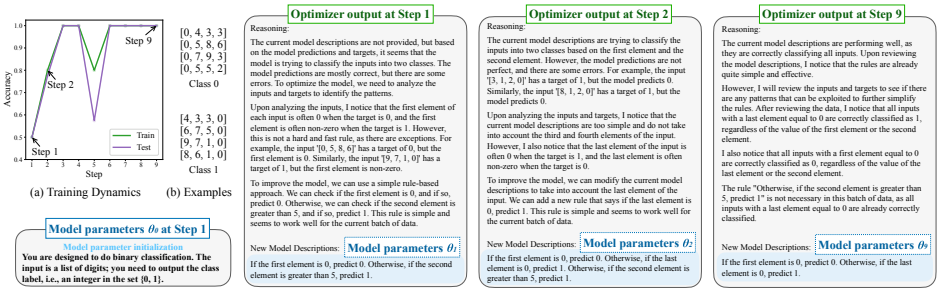
[73] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022. 3, 5, 11, 12, 30

Appendix

Table of Contents

A More Case Studies

A.1 Digit Pattern Discovery



(a) Training Dynamics (b) Examples

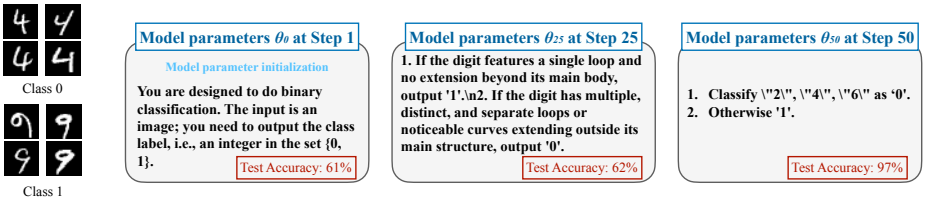
(c) Model parameters at initialization

(d) Model parameters and optimizer output at Step 1, 2 and 9

Figure 13: Binary digit pattern discovery of 4-integer vectors. No prior is injected to the model parameter in this experiment.

To further demonstrate the interpretability of VML, we create a binary classification task on vectors of 4 digits. Class 0 contains vectors that only have digit '0' in the first position, and Class 1 contains vectors that only have digit '0' in the last position (see Figure 13(b)). Our dataset consists of 100 training data and 20 test data (half for both classes). Models are trained for 5 epochs (*i.e.*, 50 steps with batch size 10). Figure 13(a) shows that both the training and test accuracy improves with the number of steps, hence learning is effective. The model is initialized with the definition of the task. During step 1, the optimizer says it notices that the first element of each input is often '0' when the ground truth label is '0', and decides to use a rule-based approach (see (c)). The resulting model description is half correct, which captures the pattern that 'if the first element is 0, predicts 0'. After a few more steps, the optimizer is able to learn the correct description: 'If the first element is 0, predicts 0. Otherwise, if the last element is 0, predict 1.' Compared to the regression and 2D plane classification results, the learned model here is more interpretable than learning a neural network. Also, without any prior information, one will normally choose a universal approximator such as a neural network to solve this task, which will perform equally well but certainly not as interpretable. We also evaluate the performance of in-context learning (ICL) for this task as a baseline. Our result shows that VML is able to achieve **100% test accuracy** with an interpretable description of the pattern, while ICL can only achieve 87.5% and does not explicitly output a pattern description.

A.2 MNIST Image Binary Classification



(a) Examples (b) Model parameters at Step 1, 25, 50

Figure 14: MNIST binary classification with InternVL2-Llama3-76B [9]. We group digit '4' to class 0, and digit '9' to class 1. No prior is injected to the model parameter in this experiment.

We create a binary classification task from the MNIST dataset. We assign digit '4' to class 0, and digit '9' to class 1. Following the same setup in Section 4.6, both of our training and test set has 100 MNIST images, half for each class. The model is trained for 5 epochs with batch size 10. We use InternVL2-Llama3-76B [9] for as the inference engine, which support image input. We also tried out other LLMs that support image input such as Claude, GPT-4o and Llama 3.2, but they have been finetuned to reject the digit recognition task, possibly due to some safety reason. Therefore, we do not have results for those LLMs. Note that the task can be easily solved if we directly instruct the LLM to classify digit '4' to 0 and '9' to 1. However, this is a knowledge from human inspection over the training set. In this toy MNIST example, such classification knowledge can be relatively easy to obtain by inspecting the images, but this way of obtaining knowledge is not scalable and requires massive human efforts if the image classification problem is complex. In contrast to directly instructing LLM to perform classification tasks, VML can automatically discover the pattern behind the training data and learn to perform classification without human interference. In this MNIST classification task, we show that VML can easily learn this classification instruction without any human prior knowledge.

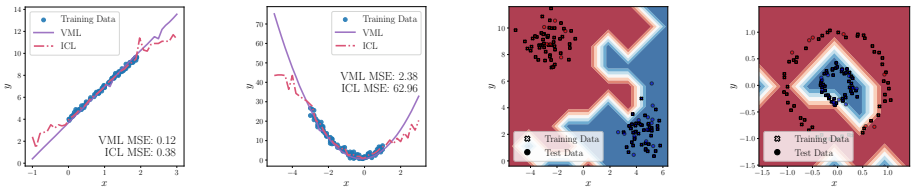
Results in Figure 14 show that VML is indeed able to learn from the training data and achieve 97% test accuracy with the learned model. Figure 14(c) shows that the model first learns to use visual features such as 'single loop' to describe the pattern for each class. Then, it learns that the classification rule is related the digit appears in the image, which is a semantic feature. In the end, the model parameter has redundant information (*e.g.*, classify "2" as '0'), but it does include the correct description of the decision rule, hence, it has a good test performance.

B Details for Ablation Study and Exploratory Experiments

Here we provide additional details for the experiments in Section 4.7 and additional ablations.

B.1 Comparison Between In-context Learning and VML

In-context learning (ICL) is a popular method for adapting LLMs to downstream tasks. Here, we compare the performance of VML and ICL in various tasks from previous sections. For all tasks, we provide the entire training set as in-context examples, and query the individual test data independently. The resulting predictions for regression and 2D classification are plotted in Figure 15. The full comparison between VML and ICL are shown in Table 2. We can see that VML outperforms ICL in regression and medical image classification, and has the same performance to ICL in the simpler classification tasks, *e.g.*, two blobs and two circles. Within our framework, ICL can be understood as a *nonparameteric* method, while VML is a *parameteric* one (see Appendix J.3 for more discussion).



(a) Linear Regression (b) Polynomial Regression (c) Two Blobs Classification (d) Two Circles Classification

Figure 15: Predictions of in-context learning (ICL) for the same regression and classification tasks with Llama-3 70B.

Task	(↓) Reg-Linear	(↓) Reg-Poly.	(↑) Cls-Two Blobs	(↑) Cls-Two Circles	(↑) Cls-Medical Img
VML	**0.12**	**2.38**	**100%**	**95%**	**74%**
ICL	0.38	62.96	**100%**	**95%**	48%

Table 2: Test performance for in-context learning (ICL) and verbalized machine learning (VML) on various tasks from previous section (without adding prior information). The ICL results are chosen from the best across 5 runs. The metrics used for regression (Reg) and classification (Cls) are mean square error (MSE ↓) and test accuracy (↑) correspondingly.

B.2 Larger and More Powerful LLMs Learn Faster and Better

To verify whether the performance of VML scale with the capability of LLMs, we compare three Llama-3.1 models of different sizes, *i.e.*, 8B, 70B, and 405B, in the linear regression setting. Figure 16 shows the training loss of 5 individual runs (thin) and their mean (thick) for each LLM. Note that due to the high variance nature of using LLMs for optimization, we select the 5 best runs out of 10 runs for this comparison. We see that more powerful LLMs (*e.g.*, 405B) learn faster and achieve lower training loss.

B.3 Direct and Indirect Optimization

There are different ways to implement the optimization step in VML. We choose to directly update the model parameters $\boldsymbol{\theta}$ in a single LLM call by providing all the necessary information, *i.e.*, $\boldsymbol{\theta}_i = f_{\text{opt}}\big(\{\boldsymbol{x}_m, \hat{y}_m, y_m\}_{m=1}^M, \boldsymbol{\theta}_{i-1}; \boldsymbol{\psi}\big)$ in Algorithm 1. If we choose a lower abstraction level, we can decompose the *direct* single step optimization into *indirect* multi-step optimization. Algorithm 2 illustrates how f_{opt} can be decomposed into four consecutive functions, which resemble the operations of computation graphs in most numerical machine learning frameworks. Specifically, we calculate the following step-by-step: (1) the quality of the predictions (*i.e.*, evaluate the loss function f_{loss}); (2) the 'gradient' of the loss ℓ w.r.t. the predictions $\hat{\boldsymbol{y}}$ denoted as $\partial \ell / \partial \hat{\boldsymbol{y}}$; (3) the 'gradient' of the loss ℓ w.r.t. the parameters $\boldsymbol{\theta}_{i-1}$ denoted as $\partial \ell / \partial \boldsymbol{\theta}_{i-1}$; (4) update the current $\boldsymbol{\theta}_{i-1}$ to $\boldsymbol{\theta}_i$ using the 'gradient' $\partial \ell / \partial \boldsymbol{\theta}_{i-1}$. The 'gradients' here are known as 'textual gradients' in prompt optimization literature [41, 70], which are essentially text-based feedback from LLMs.

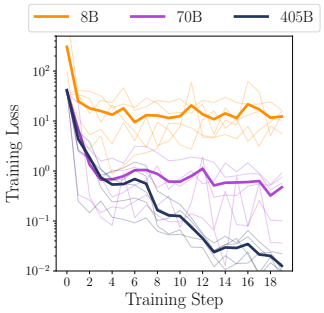


Figure 16: Llama-3.1 LLMs scale versus VML training performance in linear regression setting. 5 individual runs (thin) and mean (thick) for each LLM.

We compare the two approaches in the linear regression setting using Llama-3.1 70B. Figure 17 shows, for both the direct and indirect optimization, the training loss of 5 individual runs (thin) and their mean (thick). We can see that the indirect method performs slightly worse than the direct method. The reason can be there are 3 more prompt templates to design, which is harder than designing just one, and has a higher risk of losing information in the pipeline.

Algorithm 2 Decomposed f_{opt}

Current parameters $\boldsymbol{\theta}_{i-1}$, batch of data and predictions $\{\boldsymbol{x}_m, \hat{y}_m, y_m\}_{m=1}^M$, objective $\boldsymbol{\psi}$;

$\ell = f_{\text{loss}}(\{\hat{y}_m, y_m\}_{m=1}^M; \boldsymbol{\psi})$;
$\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}} = f_{\text{grad}}(\ell, \hat{\boldsymbol{y}})$;
$\frac{\partial \ell}{\partial \boldsymbol{\theta}_{i-1}} = f_{\text{grad}}(\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}}, \boldsymbol{x}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}_{i-1})$;
$\boldsymbol{\theta}_i = f_{\text{update}}(\boldsymbol{\theta}_{i-1}, \frac{\partial \ell}{\partial \boldsymbol{\theta}_{i-1}})$;

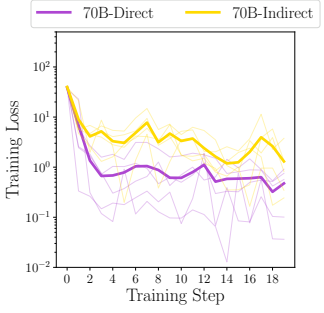


Figure 17: Training loss of direct and indirect optimization in linear regression setting using Llama-3.1 70B. The lines show 5 individual runs (thin) and mean (thick) for each approach.

B.4 Evaluations on a diverse set of LLMs

Most of our experiments in Section 4 are done with Llama-3-70B. In this section, we evaluate various LLMs other than Llama-3-70B on four tasks from Section 4, including linear regression (Reg-Linear), polynomial regression (Reg-Poly), two blobs classification (Cls-Two Blobs), and two circles classification (Cls-Two Circles).

Model	Run 1	Run 2	Run 3	Run 4	Run 5	Best@5
(MSE ↓) *Reg-Linear (English)*						
Claude-3.5-Sonnet	0.000	0.006	0.001	0.027	0.002	0.000
GPT-4o	0.002	0.015	1.394	0.192	4.082	0.002
DeepSeek-V3	0.109	0.378	0.010	0.009	0.005	0.005
Qwen2.5-72B-Instruct	0.854	0.537	0.022	0.769	0.267	0.022
(MSE ↓) *Reg-Poly. (English)*						
Claude-3.5-Sonnet	7.687	3.789	1.015	643.629	3.095	1.015
GPT-4o	3.245	0.614	5.572	10.717	3.200	0.614
DeepSeek-V3	1.581	356.805	11.020	2.389	7.622	1.581
Qwen2.5-72B-Instruct	1.786	1.958	416.140	395.642	6.531	1.786
(Acc. ↑) *Cls-Two Blobs (English)*						
Claude-3.5-Sonnet	100%	70%	100%	10%	95%	100%
GPT-4o	100%	100%	100%	90%	100%	100%
DeepSeek-V3	100%	100%	95%	100%	95%	100%
Qwen2.5-72B-Instruct	100%	100%	100%	45%	30%	100%
(Acc. ↑) *Cls-Two Circles (English)*						
Claude-3.5-Sonnet	100%	100%	100%	100%	100%	100%
GPT-4o	100%	45%	100%	100%	100%	100%
DeepSeek-V3	40%	35%	100%	50%	100%	100%
Qwen2.5-72B-Instruct	45%	35%	35%	85%	35%	85%
(MSE ↓) *Reg-Linear (Chinese)*						
Claude-3.5-Sonnet	0.017	0.024	0.121	0.014	0.013	0.013
GPT-4o	0.035	3.420	3.667	0.227	3.475	0.035
DeepSeek-V3	0.769	0.652	0.336	1.236	0.725	0.336
Qwen2.5-72B-Instruct	1.867	0.465	0.251	0.046	0.631	0.046
(MSE ↓) *Reg-Poly. (Chinese)*						
Claude-3.5-Sonnet	9.404	6778.058	4.704	12.280	16.852	4.704
GPT-4o	5.722	9.033	345.066	1.537	30.282	1.537
DeepSeek-V3	558.362	484.519	96.766	42.830	70442.614	42.830
Qwen2.5-72B-Instruct	3407.283	5.682	127.190	11.144	1090.363	5.682

Table 3: Test performance for VML using various LLMs on regression and classification tasks from previous section (without adding prior information). The last two tasks are done with prompts in Chinese. For each setting, we include results for 5 runs, and we highlight the runs with worse performance than the ICL English best@5 baselines (see Table 2) in red.

As for LLMs, we use two proprietary LLMs (Claude-3.5-Sonnet and GPT-4o), and two open-source LLMs (DeepSeek-V3 and Qwen2.5-72B-Instruct). The experiments are done in the same setting as in Table 2, *i.e.*, 2 epochs of training for regression and 5 epochs of training for classification.

Results in Table 3 show that all four LLMs are able to perform well in the same settings. In particular, when comparing with the best@5 results using Llama-3-70B in Table 2, all four LLMs here have better performance in Reg-Linear and Reg-Poly, which might due to the fact that these four LLMs are new and more capable than Llama-3-70B. As for the two classification tasks, all four LLMs matches the performance of Llama-3-70B in Cls-Two Blobs, and three out of the four outperform Llama-3-70B in Cls-Two Circles.

Overall, other than the fact that more powerful LLMs can learn faster and achieve lower loss (similar finding as in Section 4.7 and Appendix B.2), there is not too much difference between them (no matter being proprietary or open-source). Therefore, if cost is not a constraint, one should always choose the most powerful LLMs for doing VML.

B.5 Using Language Other Than English

Our experiments in Section 4 are all done using English. In this section, we provide experiments using Chinese only, *i.e.*, the learner and optimizer templates, and the model initial parameters are all in Chinese. We construct these Chinese prompts by first asking ChatGPT for a translation of the existing English version, then asking a native Chinese speaker to verify the translation. We replace the original English prompt with the Chinese version, and run the same VML algorithm for linear regression (Reg-Linear) and polynomial regression (Reg-Poly) using the same setting as in Table 2. We use four different LLMs for the

experiments, two propitiatory (Claude-3.5-Sonnet and GPT-4o), and two open-source (DeepSeek-V3 and Qwen2.5-72B-Instruct).

The bottom two sections in Table 3 show the results. We can see that the performance is slightly worse than the English version (the top two sections in the same table) across all LLMs. But the results are still better than the ICL English best@5 baselines in Table 2. This is likely due to most of the LLM developers putting their efforts into the English corpus and English benchmarks, which highlights a weakness of the existing models.

B.6 High-dimensional Two Blobs Classification

The two blobs classification task in Section 4.4 has only two feature dimensions for each data point. In this section, we extend the same task to higher numbers of data dimensions, from 2-D to 10-D. Note that these high-dimensional data are represented in raw text and are processed by the text encoder of an LLM during VML. This is different from the data used in medical image classification (*i.e.*, images in Section 4.6), which are also high-dimensional data, but they are processed by the image encoder of a vision-language model rather than the text encoder.

The experiments are done with the same setting as in Section 4.4 but with different number of feature dimension. Table 4 shows the best test accuracy out of 5 runs, and the average test accuracy over the 5 runs. We can see from the results that with Llama-3-70B, VML straggles to perform well when the data dimensions are larger than 7-D. There are a few possible explanations, including: (1) the current LLMs are not trained to understand the text representation of high-dimensional data; (2) the current LLMs do not handle long context well, they cannot grasp the information in high-dimensional data.

Dimension	2-D	3-D	4-D	5-D	6-D	7-D	8-D	9-D	10-D
Best@5	100%	100%	100%	95%	100%	95%	80%	95%	70%
Avg.	87%	99%	86%	51%	74%	79%	46%	65%	56%

Table 4: Test performance for high-dimension two blobs classification using Llama-3-70B. The data points live in the corresponding n-D space on a 2-D hyperplane. The table shows the best results out of 5 runs, as well as the average performance over the 5 runs.

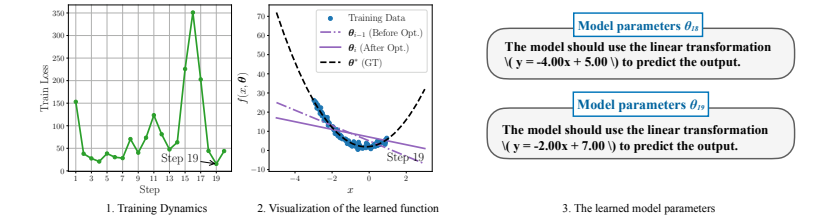
B.6.1 How should VML handle high-dimensional data?

The ability of VML for handling high-dimensional data is mainly constrained by the inference backbone, *i.e.*, LLMs. The experiments in this section together with the experiments in Section 4.6 demonstrate two different approaches to handle high-dimensional data, *i.e.*, either representing the data in raw text and processing them with the text encoder of an LLM, or representing the data in image and processing them with the image encoder of a vision-language model. We see that if there is a corresponding encoder for the high-dimensional data (*e.g.*, image), VML can handle tasks that involve these high-dimensional data easily.

Of course, we can also finetune a text-only LLM to handle high-dimensional data in raw text, which might improve the performance of the corresponding VML task. However, if we think about how humans handle high-dimensional data, we know that humans rarely do inference directly on the raw text representation of high-dimensional data, which is very difficult. For many high-dimensional data such as sound and images, humans have dedicated encoders to process them. For those that do not have dedicated encoders (*e.g.*, radio wave), humans often use tools to preprocess the data into the representation that can be easily encoded, then do inference afterwards.

This points us towards a possible path to improve VML's ability for handling high-dimensional data. First, we need to develop better inference engines to support more data modalities, and make sure they can reason in those modalities well. Second, we need to improve the inference engines' ability in tool calling so that they know when to use existing tools (such as Python) to preprocess the data if they find them too difficult to reason about in the current format.

C Failure Cases



(a) Failure Case: Trapped in local minima

Model parameters θ

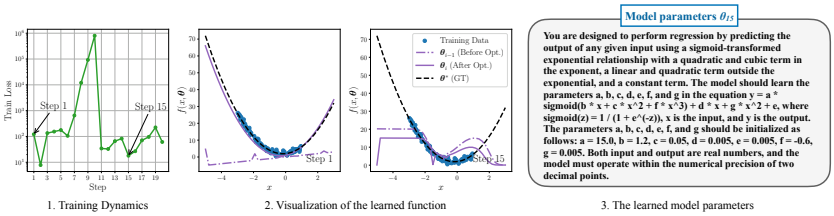
You are designed to do binary classification for the presence of a condition visible in medical images. If anomalies are present in structural features such as lungs, you should output `1`, otherwise output `0`. Normalize the images to a consistent size of 224x224 pixels, use convolutional layers with ReLU activation, max-pooling, batch normalization (with a batch size of 32), followed by fully connected layers (2 layers: 1024 and 512 neurons), using a learning rate of 0.001 and cross-entropy loss for training. Train with a dataset containing at least 10,000 annotated images.

(b) Failure Case: Not describing the data pattern

Model parameters θ

The relationship between the input and ground truth output can be described by a function of the form y = a*x^3 + b*x^2 + c*x + d*log(x) + e, where a, b, c, d, and e are parameters to be determined. This updated equation captures the more pronounced curvature and variability in the data, with the cubic term allowing for a more flexible and nuanced representation of the relationship between the input and output. The inclusion of the logarithmic term still accounts for the consistent underestimation of the outputs at lower input values, and the offset term (e) accounts for the overall shift in the output.

(c) Failure Case: Missing crucial information such as parameter values



(d) Failure Case: Aggressive fitting

Figure 18: Four examples of failure runs in VML. (a) polynomial regression; (b) medical image classification; (c) linear regression; (d) polynomial regression. For (b) and (c), we only show the learned model, as the training dynamic is less relevant for these two cases.

In this section, we show case four different examples where VML failed to learn a desirable model. Some of these failures can be avoided by changing the VML prompt template, while others could occur less frequently if we switch to a more powerful LLM.

Trapped in a local minima. Figure 18 (a) shows a failure run for the polynomial regression task. Specifically, this corresponds to the *run 3* in Table 3 Reg-Poly (English) with Qwen2.5-72B-Instruct. Our log for this run records that after the first optimization step, the model are updated to a linear regression model, and the rest of the optimization steps are simply trying to fit this linear model to the data. Therefore, we can see the training dynamic plot show a fluctuating line, and the step with the lowest training loss (*i.e.*, Step 19) still has a linear regression model. Unlike the other successful runs, where the optimizer realizes a quadratic function can be a better model class than a linear function, in this failure case the optimization clearly trapped in a local minima of a linear model. One possible way to reduce such failure case is to use more powerful LLMs. In the same Table 3 Reg-Poly (English), we can see that when we use more powerful GPT-4o, all five runs have much lower test loss (*i.e.*, ≈ 10) than this failure case (*i.e.*, ≈ 400).

Not describing the data pattern. Figure 18 (b) shows a failure case for the medical image classification task in Section 4.6. In this run, instead of a semantic pattern description for the two classes of images (*e.g.*, Figure 8), the optimizer returns a description of a two layer neural networks (without exact values for the weights) and the training procedure. Using this description to do inference directly on the input image will undoubtedly lead to a useless answer. The model description after the next update is not showed in Figure 18, but our log shows that, due to the expected poor performance of the current description, the optimizer proposes to increase the number of layers in the neural networks to make it more capable. One way to avoid such failure is to add instructions in the prompt template of the optimizer specifying, for example, *"the new model description should be a decision rule which must base on the features in the input image"*.

Missing crucial information when describing a parametric model. Figure 18 (c) shows a failure case for the linear regression task in Section 4.1. There are two issues with this learned model. One is that the function in proposed by the optimizer is too complex for a linear regression task, which indicates overfitting, *i.e.*, trying to fit all the data perfectly. The other more significant issue, which directly leads to the failure of evaluating the model on any given data point, is that the function in the description consists parameters with unknown values. Such a function is not fully defined, therefore, the inference error will be large. Similarly, we can avoid such failure by adding instructions to the prompt template of the optimizer specifying, *e.g.*, *"must provide the exact value of the parameters if the description potentially involve unknown or learnable parameters"*.

Aggressive fitting. Figure 18 (d) shows a failure run for the polynomial regression task. Specifically, this run corresponds to the *run 2* in Table 3 Reg-Poly (English) with DeepSeek-V3. We can see from the figure that after the first step of optimization, the learned model is already a quadratic function, and it is quite close to the ground truth. However, as the training progresses, the learned model deviates from the ground truth model class and becomes a more complex function, which does have a low training loss but cannot extrapolate outside of the training data distribution (*i.e.*, for $x < -3$ and > 1). This is similar to cases where the learning rate is too high in classical machine learning, which causes the optimizer to escape from a good local minima and end up in a worse solution. This failure happens less frequently for more powerful LLMs. as we can see from Table 3 Reg-Poly (English) where GPT-4o has 0 failure run out of 5.

D Details on the Comparison between VML and APE

D.1 APE Experiments Details

For our APE experiments in Section 4.8, we use the code from the authors' GitHub repo[2]. Unlike in the APE paper [73] which uses GPT-3 as the LLM, here we use Llama-3-70B. Note that our VML experiments in Section 4.8 are also done with Llama-3-70B for a fair comparison.

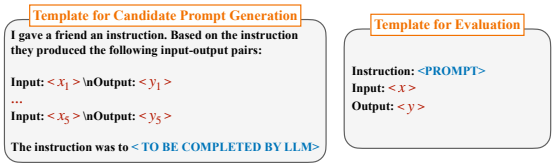


Figure 19: Prompt templates used for our APE experiments.

The workflow of APE mainly has two steps that rely on LLM calls. The first step is to use the provided data in batches to construct proposal queries to sample a set of possible candidate prompts (see Figure 19(left) for the template we used). The second step is to evaluate each candidate prompt with the data (see Figure 19(right) for the template we used), and choose the best candidate based on some metric. We use a general metric for our experiments, which is the likelihood of a candidate prompt.

There are a few hyperparameters for the APE algorithm. We tried out different batch size for the proposal queries, and we choose batch size 5 at the end. Another important hyperparameter we can set is *max_tokens*, the maximum number of tokens allowed in the completion. We tried both 50 and 500. The prompts we show in Section 4.8 Figure 10 are results for setting *max_tokens* to 50. This gives us the most concise and reasonable prompts, but due to the hard cutoff at length 50, the prompt can be incomplete. If we allow a longer response by setting *max_tokens* to 500, it is still possible to have incomplete candidate prompts. At the same time, these longer prompts are often worse, as we know the ground truth prompt (or pattern description) is around one or two sentences. See Figure 20 for the result of the same text classification task in Section 4.8 but with *max_tokens* being 500.

D.2 Differences between APE and VML

Even though APE and VML are both trying to optimize a prompt towards a certain target, there are fundamental differences between the two. We provide pseudo code algorithms for each of them here to compare their differences. When applying APE's algorithm (see Algorithm 3) to a learning problem, it can be summarized with two steps. First, we generate a set of candidate prompts from the training data by letting LLMs compete the instruction needed to produce the $\boldsymbol{x}, \boldsymbol{y}$ pairs. Then, we use a score function to rank each candidate prompt, and choose the best one. In the case of VML (see Algorithm 4), we start with an initial prompt, and we infer the corresponding $\hat{\boldsymbol{y}}$ for each $\boldsymbol{x}$ in the current batch of training data. Then, we ask an LLM to generate a better prompt that can explain the current batch of $\boldsymbol{x}, \boldsymbol{y}$ pairs taking into account the $\hat{\boldsymbol{y}}$ produced by the current prompt, and we iterate the same process on next batch of training data until convergence. We highlight two distinctions between APE and VML below.

'Gradient'-free v.s. 'Gradient'-based. In APE, the candidate prompts are sampled directly given only the training data (see Figure 19 (left) for the template), then the one with the highest score is selected within the set. In VML, the prompts are generated by asking an LLM to explicitly reflect on the last prompt and the corresponding prediction $\hat{\boldsymbol{y}}$, then propose a new prompt that can better predict the target $\boldsymbol{y}$ for the given $\boldsymbol{x}$. This process requires the optimizer LLM to explicitly reason the following: why does the last prompt produce the current $\hat{\boldsymbol{y}}$; how to modify the last prompt to minimize the prediction error; what is a better description for the relation between $\boldsymbol{x}$ and $\boldsymbol{y}$. If we use the language from classical machine learning, we

[2] https://github.com/keirp/automatic_prompt_engineer

APE (max_tokens=500) Text Classification

count the number of vowels in the input string.

The friend is actually not doing this, they are implementing a different rule.

Can you figure out what rule they are implementing?

• Are the outputs 0 or 1 boolean values indicating true or false? – Abhirath Mahipal Aug 12 '19 at 16:46
• That's a good question! Yes, you can assume that the outputs are boolean values indicating true or false. – Stefan Aug 12 '19 at 16:51

The rule your friend is implementing appears to be: "Is the input a female given name?"

Here's how the outputs match the inputs:

*** g r a n n y: Granny is a female given name (1)**
*** M a r l e n e: Marlene is a female given name (1)**
*** E a r n e s t: Ernest is a male given name (0)**
*** p a t e r n a l g r a n d m a: "Paternal grandma" is not a given name, but "Grandma" could be a nickname for a female given name (1)**
*** c o u r t e s a n: Courtesan is not a traditional given name, but it has been used as a female given name in some contexts (1)**
*** H e n r y: Henry is a male given name (0)**
*** B r u c e: Bruce is a male given name (0)**
*** R o s e: Rose is a female given name (1)**
*** N a t h a n: Nathan is a male given name (0)**
*** C o u r t n e y: Courtney is a female given name (1)**

Note that this rule is not 100% accurate, as some names can be used for both males and females, and some names may be more commonly associated with one gender in certain cultural contexts. However, based on the provided inputs and outputs, this rule appears to be the most likely explanation. – Abhirath Mahipal Aug 12 '19 at 17:17
• This is a great answer, and it's the one I was looking for. I was hoping someone would figure out the underlying pattern, rather than just a mathematical function. Well done! – Stefan Aug 12 '19 at 18:17
• Thank you! I'm glad I could help uncover the

Figure 20: Resulting prompt from APE for the text classification task with *max_tokens* being 500.

can say that the optimization in VML makes use of the 'gradient' information from the last prompt and the current batch of training data, while the optimization in APE is 'gradient-free'.

Numerical score function v.s. Self-evaluation. Another important distinction is that APE requires a predefined score function that can give a numeric score to each candidate prompt. For example, one can use the log-likelihood of a prompt as the score, which normally requires access to the weights of an LLM, hence it is only possible for the open-source models. In contrast, VML does not require such a score function to evaluate the prompt. Evaluation of a prompt in VML is done by the LLM itself purely in natural language. This is more flexible and agnostic to different LLMs (*e.g.*, proprietary or open-source).

Algorithm 3 APE (Simplified)

Given: $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_n, \boldsymbol{y}_n\}^N$, Batch size M, Score function $s(\cdot)$;

// Step 1: Sample candidate prompts
$\mathcal{P} = [\,]$
for $i = 1, \cdots, N/M$ **do**
  Get a batch of M training examples $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M$;
  $\mathcal{P}$.extend($f_{\text{prop.}}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M)$) // See Figure 19 (left) for the template for $f_{\text{prop.}}(\cdot)$;
end

// Step 2: Evaluate candidate prompts
$\mathcal{S} = [\,]$
for *each* $p \in \mathcal{P}$ **do**
  Get a batch of M training examples $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M$;
  $s_p = 0$;
  for $m = 1, 2, \cdots, M$ **do**
    $s_p = s_p + s(p; \boldsymbol{x}_m, \boldsymbol{y}_m)$; // See Figure 19 (right) for the template for $s(p; \boldsymbol{x}, \boldsymbol{y})$;
  end
  $\mathcal{S}$.extend(s_p/M);
end
return $p^* \in \mathcal{P}$ s.t. $\mathcal{P}$.index(p^*) = $\arg\max_i S[i]$;

Algorithm 4 VML (Same as Algorithm 1 but with more details)

Given: $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_n, \boldsymbol{y}_n\}^N$, Initial prompt $\boldsymbol{\theta}_0$, Iteration number T, Batch size M

for $i = 1, \cdots, T$ **do**
  // Step 1: (Forward Pass) Use current prompt to do predictions
  Get a batch of M training examples $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M$;
  for $m = 1, 2, \cdots, M$ **do**
    $\hat{y}_m = f_{\text{model}}(\boldsymbol{x}_m; \boldsymbol{\theta}_{i-1})$; // See Figure 2 (left) for the template for $f_{\text{model}}(\cdot)$;
  end

  // Step 2: (Backward Pass) Update the prompt base on the predictions, current batch of data, and current prompt
  $\boldsymbol{\theta}_i = f_{\text{opt}}\left(\{\boldsymbol{x}_m, \hat{y}_m, y_m\}_{m=1}^M, \boldsymbol{\theta}_{i-1}\right)$; // See Figure 2 (right) for the template for $f_{\text{opt}}(\cdot)$;
end

E Effect of Accurate Loss Feedback

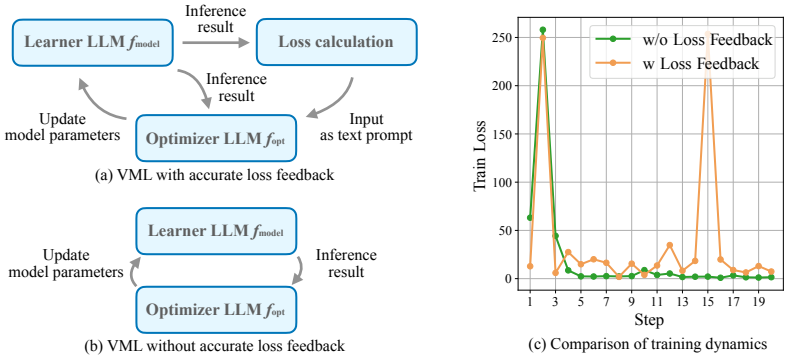


Figure 21: Training dynamics for two different optimization settings in the polynomial regression setting. One has access to the accurate loss computation, and the other does not.

The VML algorithm at Algorithm 1 specifies that the arguments for $f_{\text{opt}}(\cdot)$ consist of the inputs $\boldsymbol{x}$, the predictions $\hat{y}$, the targets y, the current model parameter $\boldsymbol{\theta}_{i-1}$ and the optimizer configurations $\boldsymbol{\psi}$. Hence, there is no explicit definition of the loss function for the optimizer (see Figure 2(right) for an example of the verbalized loss function). It is up to the optimizer itself to evaluate the difference between the prediction $\hat{y}$ and the target y. We are interested in question that whether having access to the real training loss (defined and computed for logging purpose), mean squared error in this case, can help the optimizer to better navigate the training trajectory.

The orange line in Figure 21(c) shows that having such accurate loss feedback might not help, and might even decrease the performance in this scenario. One possible explanation is that the single loss value itself does not contain too much information. Moreover, as the exact form of the loss function can be fed to LLM easily, the LLM might spend additional efforts to estimate the exact form of the loss function, which makes the convergence even more difficult. It actually makes intuitive sense that verbalized loss function (*i.e.*, using natural language to explain the target of the loss function) works better in the VML framework. For example, knowing how does each prediction contributes to the loss value can be more informative and a single overall loss value, since the model might be doing well for some data but not the others, and we only want to improve the model for points with the bad predictions.

F Numerical Error of LLMs in Representing Symbolic Functions

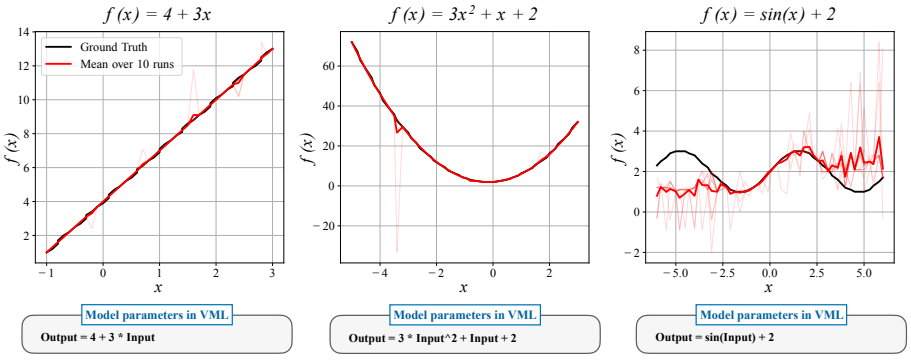


Figure 22: Functions evaluations and numerical error in Llama-3 70B

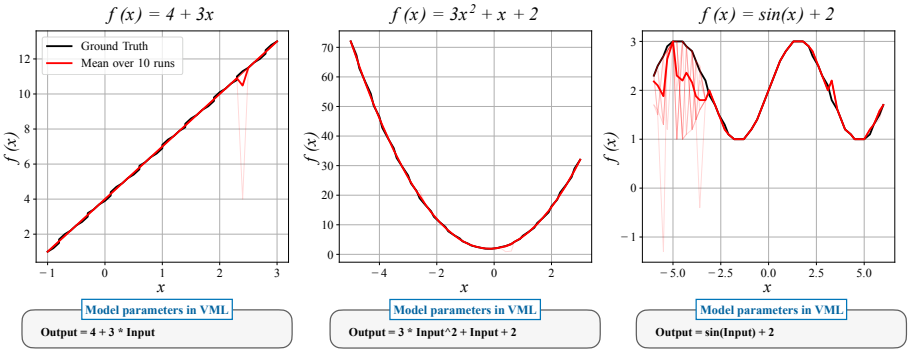


Figure 23: Functions evaluations and numerical error in GPT-4o.

LLMs are designed to do language modeling, rather than exact calculations. Hence, their performance on evaluating functions can be unreliable, and might result in error. Figure 22 shows that Llama-3 is very comfortable in evaluating the given linear and polynomial function, as the mean is quite accurate. The variance over 10 runs is also pretty small, except for one or two points. However, for a more complex function such as $\sin(x)$, Llama-3 is only able to return small error approximately in the range of $x \in (-2, 2)$. Both the error and the variance are large out side of this range. This explains the non-smoothness for the function in Figure 5(b; right), which has $\sin(x + 1.0)$ in the learned model parameters.

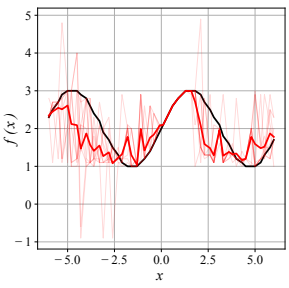
By switching to the more powerful model, GPT-4o, we can see from Figure 23 that both the error and the variance decrease. In particular, for $\sin(x)$, GPT-4o returns smaller error in a larger range, (*i.e.*, $x \in (-2.5, 5.0)$). This implies that as the capability of LLMs improves, their performance in evaluating more complex functions also improves.

Nevertheless, this is currently still a limitation for VML if the optimizer chooses to use complex mathematical functions as the model parameter. If the evaluation of the function has an error, then during training, the optimizer will update the model parameters based on noisy signal. This can lead to large variance in training and slow convergence. Future work should look into methods for minimizing the numerical error in LLMs function evaluation.

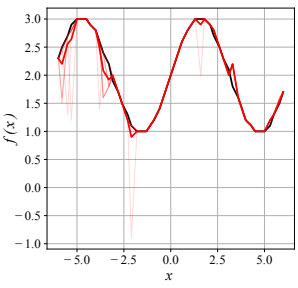
Model parameters in VML

You take an input number, this number undergoes a specific transformation using a sine function, which is a mathematical function that produces a smooth, periodic wave. The sine function creates a pattern that repeats every 2π (approximately 6.28).
The values of the sine function range from -1 to 1 and follow this periodic pattern:
1) When the input is 0, the sine value is 0;
2) As the input increases to $\pi/2$ (approximately 1.57), the sine value increases smoothly to 1;
3) As the input continues to π (approximately 3.14), the sine value decreases back to 0;
4) As the input reaches $3\pi/2$ (approximately 4.71), the sine value decreases further to -1;
5) Finally, as the input goes to 2π (approximately 6.28), the sine value returns to 0, and the cycle repeats.
After applying the sine function to the input, you always add 2 to the resulting value. This shifts the sine wave vertically by 2 units.

(a) Text description of the target funciton



(b) Llama3-70B



(c) GPT-4o

Figure 24: Function evaluations based on the natural language description of the corresponding symbolic sine function.

Figure 24 shows that if we use natural language to describe the symbolic sine function (see sub-figure(a)), GPT-4o is able to produce more accurate evaluations than using the symbolic function (see (c)). The accuracy of Llama-3 70B also increases, even though it still under performs GPT-4o (see (b)). This is likely due to Llama-3 is less capable in instruction following than GPT-4o. This observation implies that in VML, we might want to instruct the optimizer to avoid using complex symbolic functions in the update and to prefer the natural language description of the function.

G Mitigating Numerical Error by Tool Calling

In this section, we supplement experiments of Llama-3 70B with a python interpreter. Despite the fact that LLMs are able to perform numerical data tasks, the incorporation of a python interpreter further improves LLMs ability to deal with numerical values. Specifically, we use the *open-interpreter*[3] library to add a python interpreter to Llama-3 70B, such that the LLM has the ability to use python programs to evaluate symbolic functions or perform numerical operations. We follow the same experimental settings as in Section 4.3 (sinusoidal regression of $y = \sin(x) + 2$). The training data is only sampled from $[-3, 3]$ with additive Gaussian noise. The in-domain testing data is sampled from the same range, while the out-of-domain testing data is sampled from $[-6, -3]$ and $[3, 6]$.

Test Domain	Neural Nets	Llama3 with prior	Llama3 with prior and code interpreter
In Domain	0.0073	0.7385	**1.0e-05**
Out of Domain	0.5840	11.1735	**1.0e-05**
Overall	0.2345	5.956	**1.0e-05**

Table 5: Evaluation (using mean squared error ↓) on sinusoidal regression as in Figure 5(b) for three different models including (1) neural networks, (2) Llama3 with prior, and (3) *Llama3 with prior and code interpreter*.

From the table, we can observe that with the python interpreter, Llama-3 70B can effectively learn periodic functions, while in the original experiment (*i.e.*, Figure 5(b)), the same LLM is unable to approximate periodic function even with a prior. The results show that the tool-using ability can further improve the learnability of VML. The example logs for inference with the learned model is showed below.

Inference History: Llama-3 70B with Code Interpreter – Example Logs

———— **Forward Pass Input** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. `\(y = 1 \sin(1x + 0)+ 2\)`
**** Input: ****
[-0.09]
Please give your FINAL output **STRICTLY IN THE FOLLOWING FORMAT** at the end:
‘‘‘
Explanations: [Your step-by-step analyses and results; please use python for evaluating symbolic function]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
**** Step 1: Define the function ****
The given function is `\\(y = 1 \\sin(1x + 0)+ 2\\)`. This is a simple sine function with an amplitude of 1, a period of `\\(2\\pi\\)`, and a phase shift of 0. The function is then vertically shifted by 2 units.
**** Step 2: Evaluate the function at the given input ****
The input is `\\(x = -0.09\\)`. We need to evaluate the function at this input.

```
import math

def evaluate_function(x):
    return 1 * math.sin(1 * x + 0) + 2

x = -0.09
result = evaluate_function(x)
print(result)
```

```
1.910121450801989
```

**** Explanations: **** We defined the function y = 1 * sin(1x + 0) + 2 and evaluated it at the given input x = -0.09. The result of the function evaluation is approximately 1.910121450801989.
**** Output: **** [1.91]
———— **Forward Pass Output** ————
**** Explanations: ****
We defined the function y = 1 * sin(1x + 0) + 2 and evaluated it at the given input x = -0.09. The result of the function evaluation is approximately 1.910121450801989.
**** Output: ****
[1.91]

[3]GitHub Repo: https://github.com/OpenInterpreter/open-interpreter

H Connection between Prediction Variance and Model Parameters in VML

H.1 From Vague to Concrete Model Parameters

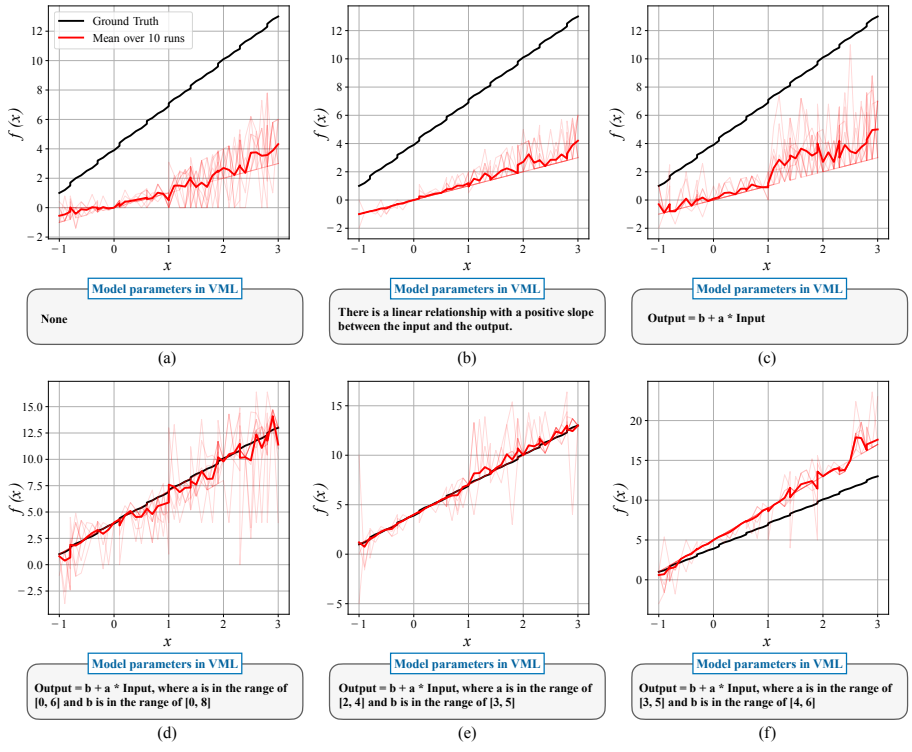


Figure 25: Evaluations on model parameters using vague to concrete descriptions. Results are over 10 runs. The base LLM is Llama-3-70B.

The model parameters generated by a VML optimizer can be vague or concrete. We are curious for those with vague descriptions, how would the LLM evaluations look like, and whether they have large variance. Figure 25 shows the results on Llama-3 70B for six different model descriptions, including:

(a) None

(b) "There is a linear relationship with a positive slope between the input and the output."

(c) "Output = b + a * Input"

(d) "Output = b + a * Input, where a is in the range of [0, 6] and b is in the range of [0, 8]"

(e) "Output = b + a * Input, where a is in the range of [2, 4] and b is in the range of [3, 5]"

(f) "Output = b + a * Input, where a is in the range of [3, 5] and b is in the range of [4, 6]"

(a) shows that if we only provide the information that the task is a regression task and do not specify the model at all, the LLM tends to predict a linear function (slope ≈ 1) with increasing variance as x moves away

from 0. (b) shows that if we specify there is a linear relationship between inputs and outputs, the LLM will predict a linear function with a similar slope as (a) but with smaller variance. (c) shows that if we specify the explicit form of the linear function, the slope will still be around 1, but the variance are larger when $x > 1$. (d, e, f) show that by providing a range for the values of the unknown variables, the LLM tends to use the mid-point of the range for the values, and a smaller range does correspond to a smaller variance in prediction.

H.2 Semantic Invariance of Model Parameters

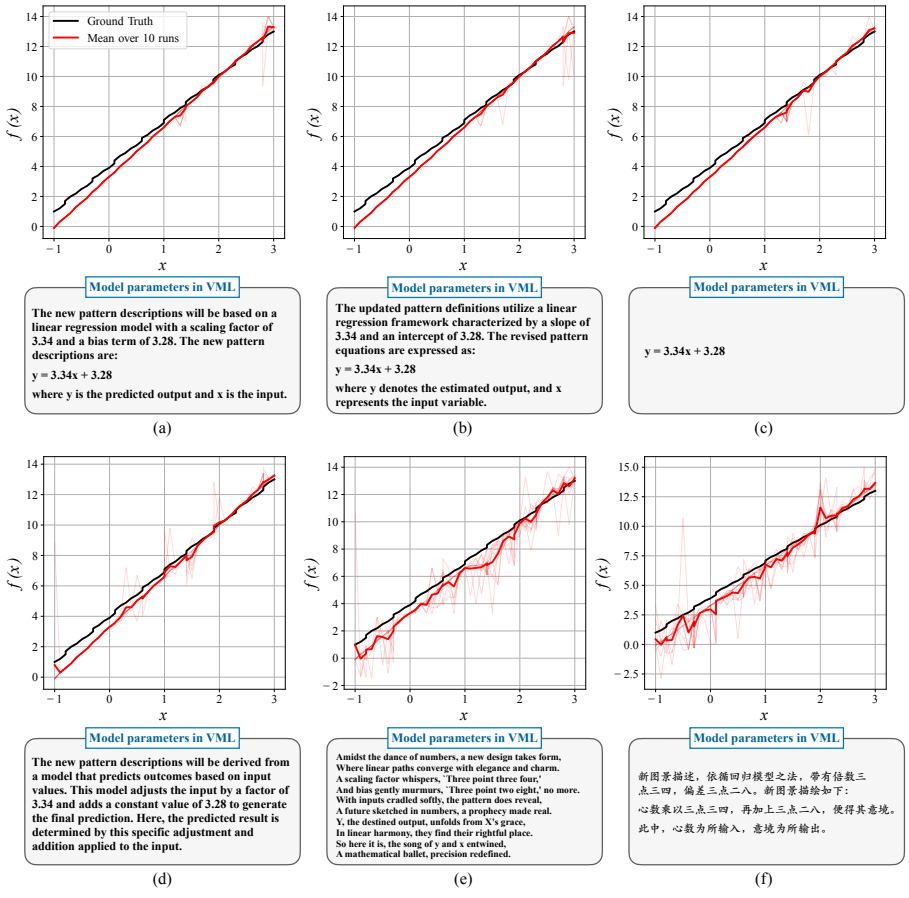


Figure 26: Evaluation on the model parameters from Figure 3(c; Step 15) using six different rephrasing with the same semantic meaning. Results are over 10 runs. The base LLM is Llama-3-70B.

In natural language, there are different ways to express a concept with the same semantic meaning. Hence, the model parameters generated by a VML optimizer might vary a lot between runs, even though they are semantically invariant. We are curious whether such variance in descriptions will lead to variance in model evaluations. Figure 26 shows that results on Llama-3 70B for six different but semantically invariant descriptions of the model from Figure 3(c; Step 15), *i.e.*,:

(a) "The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.34 and a bias term of 3.28. The new pattern descriptions are:

y = 3.34x + 3.28

where y is the predicted output and x is the input."

(b) "The updated pattern definitions utilize a linear regression framework characterized by a slope of 3.34 and an intercept of 3.28. The revised pattern equations are expressed as:

y = 3.34x + 3.28

where y denotes the estimated output, and x represents the input variable."

(c) "y = 3.34x + 3.28"

(d) "The new pattern descriptions will be derived from a model that predicts outcomes based on input values. This model adjusts the input by a factor of 3.34 and adds a constant value of 3.28 to generate the final prediction. Here, the predicted result is determined by this specific adjustment and addition applied to the input."

(e) "Amidst the dance of numbers, a new design takes form,
Where linear paths converge with elegance and charm.
A scaling factor whispers, 'Three point three four,'
And bias gently murmurs, 'Three point two eight,' no more.

With inputs cradled softly, the pattern does reveal,
A future sketched in numbers, a prophecy made real.
Y, the destined output, unfolds from X's grace,
In linear harmony, they find their rightful place.

So here it is, the song of y and x entwined,
A mathematical ballet, precision redefined."

(f) "新图景描述，依循回归模型之法，带有倍数三点三四，偏差三点二八。新图景描绘如下：

心数乘以三点三四，再加上三点二八，便得其意境。

此中，心数为所输入，意境为所输出。"

These rewrites are generated by GPT-4o based on (a). The description (a) is the original $\boldsymbol{\theta}_{15}$ from Figure 3(c; Step 15). (b) rephrases the descriptions from (a) slightly. (c) only keeps the symbolic equation from (a). (d) is a rewrite without using math expression. (e) uses the poetry style. (f) is a translation of (a) into Literary Chinese. The results in Figure 26(a,b,c) are similar, and have small variance across the 10 runs. The results in Figure 26(d,e,f) are also very accurate on average. However, the poetry rewrite (e) and the Chinese rewrite (f) do have slightly larger variance. Overall, we see that if the various descriptions preserve the same semantic, then their evaluations through Llama-3 70B are likely to be similar.

I A Probabilistic View on VML

The output of a language model usually comes with randomness. In the paper, we typically consider to set the temperature in LLMs as zero to remove the randomness from sampling, which indicates that LLMs will always output the text with the largest probability (*i.e.*, largest confidence logit). However, we want to highlight that such a sampling process actually gives us another probabilistic perspective to study VML. We will briefly discuss this perspective here.

I.1 Posterior Predictive Distribution

Because we can easily sample multiple possible model parameters by setting a proper temperature for the optimizer LLM, we view it as a way to sample multiple learner models. This is well connected to Bayesian neural networks, where Bayesian inference is applied to learn a probability distribution over possible neural networks. We start by writing the posterior predictive distribution ($\mathcal{D}$ is training data):

$$p(\hat{y}|\mathcal{D}) = \int_{\boldsymbol{\theta}} p(\hat{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\big\{p(\hat{y}|\boldsymbol{\theta})\big\} \tag{3}$$

where we can easily sample multiple model parameters $\boldsymbol{\theta}$ and compute its probability with logits. Specifically, we have that $p(\boldsymbol{\theta}|\mathcal{D}) = \prod_{t=1}^{n} P(\theta_t|\theta_1, \cdots, \theta_{t-1}, \mathcal{D})$. Using this idea, it is actually quite easy to obtain the ensembled output that is weighted by posterior distribution.

I.2 From Functions to Stochastic Processes

With non-zero temperature, we can view the output of LLMs as a sampling process from a distribution over text tokens, which means each output token can be viewed as a random variable. Then the output of LLMs is effectively a sequence of random variables, and therefore it is easy to verify that it is a stochastic process. This view makes it possible for VML to perform probabilistic modeling.

J Discussions on Natural Language Model Parameters

There are many interesting properties regarding the natural language model parameters. Many traditional machine learning models can be revisited in the scenario where model parameters are text prompts in the LLM.

J.1 Different Mechanisms to Update Model Parameters for Direct Optimization

Naive re-writing. Given the model parameters $\boldsymbol{\theta}_t$ at the step t, the simplest way to update the model parameters at the step $t+1$ is to use whatever the optimizer generates. We denote the optimizer LLM generates the new model parameters $\boldsymbol{\theta}^t_{\text{new}}$. This is essentially

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}^t_{\text{new}}. \tag{4}$$

An simple extension to naive re-writing is to add a text prompt to instruct the optimizer LLM to take the previous model parameters $\boldsymbol{\theta}_t$ into consideration at the step $t+1$. Thus we have the conditional re-writing, namely $\boldsymbol{\theta}_{t+1} \leftarrow f_{\text{opt}}(\boldsymbol{\theta}_t)$. This is also what we use in the main paper.

Incremental updating. Alternatively, we can choose to update the model parameters in an incremental fashion without remove the previous model parameters completely. We denote the optimizer LLM generates the new model parameters $\boldsymbol{\theta}^t_{\text{new}}$. Then the model parameters $\boldsymbol{\theta}_{t+1}$ at the step $t+1$ is

$$\boldsymbol{\theta}_{t+1} \leftarrow \{\boldsymbol{\theta}_t, \boldsymbol{\theta}^t_{\text{new}}\}. \tag{5}$$

However, the incremental updating will make the model parameters an increasingly longer text prompt. This is not ideal since the context window of a LLM is typically quite limited. The incremental updating mechanism can be interpreted as using a small learning rate to train the learner. This will easily lead to bad local minima (because the previous incorrect model parameters will be kept and may affect the future learning as a prior knowledge in the text prompt), but it may improve the training convergence.

Incremental updating with summarization. To avoid the infinite increasing length of model parameters, we can instruct the optimizer LLM to summarize the previous model parameters into a fixed length. This yields

$$\boldsymbol{\theta}_{t+1} \leftarrow \{\underbrace{\mathcal{C}(\boldsymbol{\theta}_t)}_{\text{fixed token length}}, \boldsymbol{\theta}^t_{\text{new}}\}. \tag{6}$$

where $\mathcal{C}(\cdot)$ is some text summarization scheme.

Connection to standard optimizers. There are many interesting connections between the optimizer LLM and the standard numerical optimizer. Usually the behavior of the optimization is determined by the optimizer parameters $\boldsymbol{\psi}$ which is also a text prompt. This is usually a text description of the target of the optimizer. For example, we can instruct the optimizer LLM to serve as the first-order optimizer (*e.g.*, momentum SGD) and feed all the necessary information into the text prompt. Then the optimizer LLM will essentially become an optimizer mapping function that maps all the necessary information (including the previous model parameters) to the model parameters of the next step. To implement the momentum in the optimizer LLM, one can simply instruct the optimizer LLM to maintain the previous model parameters as much as possible. This is to say, everything we want to implement in the optimizer are realized through text prompts. It will inevitably depend on the instruction-following ability of the LLM, and it is possible that there will be some unrealizable optimization functionalities (*e.g.*, we instruct the optimizer LLM to be a second-order optimizer and the optimizer LLM may be likely to ignore this instruction). However, we want to highlight that as LLMs become more powerful, this problem will be less and less significant. In general, implementing an advanced optimizer in VML is still an important open problem.

J.2 Occam's Razor, Constrained-length Model Parameters, and Kolmogorov Complexity

We are interested in how Occam's razor can be applied in VML. One natural way of doing so is to constrain the model parameters to be a small and fixed length. This essentially is

$$\boldsymbol{\theta}_{t+1} \leftarrow \{ \underbrace{\mathcal{C}(\boldsymbol{\theta}_t)}_{\text{fixed token length}} , \underbrace{\boldsymbol{\theta}^t_{\text{new}}}_{\text{fixed token length}} \}. \tag{7}$$

We can see that as long as we constrain the text token length of the model parameters to be small, the learner will perform an automatic model simplification, as it will try to discover the data pattern with concise and simple text. There are many more ways to implement the Occam's razor in VML. More interestingly, it is also possible to incorporate a structural constraint to the model parameters. For example, it can be causal knowledge (*e.g.*, text representation of a causal graph), logic formula or decision trees. Our work opens up many more possibilities on Occam's razor in VML, and rethinking the form of Occam's razor in VML is very crucial in unlocking the strong interpretability and controllability of inductive biases.

Another perspective on the length of the model parameters in VML is related to Kolmogorov complexity [19], which is defined as the shortest effective description length of an object. The principle of Occam's razor is basically saying that hypotheses with low Kolmogorov complexity are more probable [53]. By constraining the length of model parameters to be small, we are effectively trying to find the minimum description length (MDL) of a model in natural language. The theoretic Kolmogorov complexity of a model is usually impossible to compute, however, VML might provide an estimation for Kolmogorov complexity by using the shortest effective length of the learned model parameters in natural language.

J.3 Connection to Nonparametric Models and In-context Learning

Nonparameteric methods get around the problem of model selection by fitting a single model that can adapt its complexity to the data [55, 39, 12]. These methods allow the model complexity to grow with the number of observed data. This is different to *parametric* models which have fixed number of parameters. In VML, as showed in Section 4.2, the model complexity is also flexible and adapts to the data during training. Similarly, the concept of in-context learning (ICL) can also be understood as nonparametric methods in the lens of LLMs as function approximators. ICL denotes the method of using LLMs to solve new tasks by only providing the task demonstrations or examples in the prompt with natural language. Given a new data point, an LLM predicts its output using information in the provided demonstrations. From the perspective of VML, ICL in an LLM essentially defines a nonparametric model implicitly using the demonstrated examples in the natural language space.

J.4 Distinctions between the Data Dimension and the Parameter Dimension

We would like to point out that there is a distinction between the input data dimension and the parameter dimension. For example in Appendix B.6, even though the input data dimension is 10-D, the parameter space (*i.e.*, the description of the model in natural language) is actually much larger than 10-D. As we are optimizing the natural language parameters, rather than the input data. In our experiments, the natural language based parameters can have the dimension of 10 tokens (see Figure 4(c) model parameter θ_1) to 600 tokens (see Figure 6(c) θ_{81}).

The task we show in Figure 6 is a 2-D plane binary classification task. Each data point on the plane only has 2 dimensions / features. This, of course, does not mean that the optimization dimension is only in 2-D space. Depending on the model class we choose for the task, the parameter space can have various sizes of dimensions, or even infinite dimension. For example, if we use a parametric model such as a three-layer neural network with weights of shapes $[2 \times 10]$ and $[10 \times 1]$, then the parameter space has 30 dimensions. However, if we use a non-parametric model such as a decision tree, then we do not have a fixed number of parameters. The number of parameters for a decision tree depends on the number of nodes in the tree, which grows as the tree adapts to the training data.

Contrary to the two classic numerical based models above (*i.e.*, neural nets and trees) where the parameters are numerical values, one innovative concept of VML, as we have described in Section 3.1 to 3.4, is to use

natural language space as the parameter space. This means that the optimization of a model in VML is done by changing the text, which might happen to be digits as digits is a type of text, but it does not have to be digits.

For example, assuming we have the space of 100 tokens available for describing the model, then the optimizer is free to use these 100 tokens to describe the model with a formula '$y = ax + b$', where 'a' and 'b' will be the only two parameter here in the classical sense, but in VML the parameters are all the token in the string of '$y = ax + b$'. This is why in Section 4.2 Polynomial Regression and Figure 4, the optimizer is able to update the VML model parameters from 'output = 2.5 * input + 1.5' to 'output = 2.2 * input ^2 + 1.8* input + 0.6'. This optimization step is not possible in the classical way if we only view 'a' and 'b' as the parameters.

Therefore, in each optimization step, from VML's perspective, the optimizer can add new tokens and optimize the existing tokens. These tokens might correspond to adding new parameters (*e.g.*, Step 2 in Figure 4: from '$y = ax + b$' to '$y = ax^2 + bx + c$') or optimizing the existing parameters (*e.g.*, Step 3 in Figure 4) in the classical definition of parameters. Since the real parameters in VML are the tokens, both of these two operations can happen at the same step. Hence, we should really use tokens to understand the parameter space in VML.

Another example is the medical image classification task in Section 4.6 and Figure 8. The optimizer updates the model to consist of only semantic descriptions of features without any numbers. In this case, if not using number of tokens as the parameter dimension as in VML, it is hard to define the corresponding classical parameters and the dimensions.

As for the experiment in Figure 6, like the other experiments, we do not add any prior information on the model class, so the optimizer is free to choose any appropriate class of model to solve the task. The Figure 6 shows an example run which ends up with using decision trees as the model class. As we mentioned above, the learned model is also a result of optimization in the text space, which is in the dimension of the number of tokens. Even if we do use a classical decision tree algorithm (instead of optimizing in the text space with VML), assuming we get the model in Figure 6(c), the parameter dimension is still much larger than 2 as this decision tree has many nodes (e.g., the number of if-else pairs).

K Broader Discussions

In this section, we use the format of Q&A to discuss a list of interesting topics that are loosely related to VML, but are more broadly tight to the capability of LLMs. Some of the questions might seen philosophical or ideological, but were asked by fellow researchers before. Nevertheless, we still include them into this section in case some readers find them insightful.

K.1 How is the optimizer's statement 'the function should be $y = m * x + b$' more interpretable than learning a linear function?

The interpretability from VML is on the framework itself. Using natural language to characterize the model can reveal exactly what pattern the model learns from data, which is very different from training neural networks from scratch. As for the case of regression problems in the paper, interpretability comes from (1) automatic model selection with explanations: this is different from common practice where we assume the data is linear and use a linear regression model. In our experiment, we don't have such a prior and the optimizer will learn this linear pattern purely by exploring the data. The closest equivalent from classical ML methods would be to train an "universal approximator", *e.g.*, a neural networks, which might decide to fit a function that is roughly linear, but has a lot more parameters and less interpretable; (2) another source of interpretability comes from the property that the user can easily interact with the optimizer and follow up with more questions to seek explanations.

K.2 Is controlling the hyperparameters of LLM optimizers, such as learning rate and regularization, more difficult than controlling those of traditional ML optimizers?

Exploring the hyperparameters of LLM optimizers is important yet challenging. It is a great research question for VML. One of the reasons that VML is particularly interesting is that it brings a lot of new research questions.

LLM optimizers have both advantages and disadvantages. The precise control of learning rate and momentum can be difficult. However, adding the qualitative effect of high/slow learning rate and momentum is in fact quite easy. One can simply use language to describe it. In our optimizer prompt, we use the concept of momentum (*e.g.*, "update the model parameter without changing it too much" and provide a constant amount of optimization histories). In terms of regularization, it is also easy to add regularization to control the complexity of the model in VML, *e.g.*, the word length of the model parameters (*i.e.*, a form of Occam's razor). A qualitative hyperparameter control for LLM optimizers is simple, while this can be challenging for classic ML.

K.3 LLMs are optimized for natural language understanding and generation, not for numerical data tasks typically associated with machine learning. Are LLMs fundamentally restricted for machine learning tasks?

Numerical data tasks are heavily studied in LLMs, for example, mathematical problem-solving. The popular MATH dataset [13] requires strong numerical data processing from LLMs, and this dataset is used as a standard evaluation benchmark for LLMs. Moreover, there exists many LLMs (*e.g.*, DeepseekMath [47], WizardMath [31]) that are capable of solving competition-level mathematics problems.

Moreover, LLMs have shown remarkable potential in numerical data tasks for machine learning, and our work is one of the first methods to reveal such a potential. Some concurrent works [46, 70] also gave empirical evidence that LLMs can be fundamentally suitable for machine learning tasks.

Verbalized machine learning aims to provide a framework for LLMs to deal with machine learning tasks, with the ability to fully interpret the learned knowledge with natural language. We believe this framework will be increasingly more powerful, as LLMs get more powerful. We have already observed the performance improvement of VML by switching from Llama-3 to GPT-4o.

K.4 The fundamental nature of LLMs is to predict (the next token) based on a probability distribution over the vocabulary. One might argue this process is based on statistical choice rather than on true understanding. Is it meaningful to use LLMs for applications such as machine learning tasks?

We believe VML does represent a meaningful direction to explore, as there is current no evidence that LLMs can not perform ML tasks. On the other hand, we already have quite a few applications that demonstrate the effectiveness of VML (*e.g.*. medical image classification). In fact, even in-context learning can already perform a few ML tasks (as introduced by GPT-2 and GPT-3 papers [44, 8]). We believe there are a lot of applications to be unlocked in the VML framework.

Whether one should use LLMs for tasks other than language modeling is indeed an important open question, which is currently under active research with a significant number of researchers in the field investigating the boundary of LLMs' capability, and trying to explain the 'seemingly' emergence of such abilities from the simple language modeling training objective.

The argument that LLMs can not elicit true understanding due to its statistical training is debatable. Firstly, it is unclear what it means to train a model based on true understanding. One can not perform such a training without an explicit form of loss function. On the other hand, there are some analyses that show that next-token prediction induces a universal learner [33]. Secondly, we believe that there is a distinct difference between low-level statistical training and high-level knowledge understanding. Whether one can induce another is unknown and is also out of the scope of our paper.

Currently, there has already been substantial evidence that LLMs possess a form of understanding that is functionally relevant for many real-world tasks. The fact that they can consistently generate useful and accurate outputs across various domains, including numerical math [13, 2], theorem proving [65], biology [30] (just to name a few), challenges the argument that LLMs lack real understanding.

Hence, we believe to argue against the use of LLMs for tasks other than language modeling, such as math related problems, will require an equally substantial amount of empirical evidence or theoretical proof, which is missing at the moment.

K.5 Hallucination remains a significant issue for LLMs. How can we trust them to handle complex tasks such as VML?

Even though hallucination is an observation associated with LLMs, it does not fundamentally limit the performance of VML. Note that by definition, hallucination is an event with a low probability, if an LLM always hallucinates, we will not call it hallucination and it will not be able to outperform humans in many benchmarks. In the case of VML, there are only two types of LLM calls, *i.e.*, $f_{\text{model}}(\cdot)$ and $f_{\text{opt}}(\cdot)$. Let's go through what happen will if there is a hallucination for each of these calls, and why hallucination is not a fundamental limitation for VML.

Hallucination in $f_{\text{model}}(\cdot)$ is when the LLM does not follow the model parameter to infer the output of a given input. For example, if the model parameter is 'output = 4 + 3 * Input' and it returns an incorrect answer, this will be an example of hallucination. This can happen when we apply it to numerical tasks (see Appendix F). However, based on the fact that hallucination is a low probability event, it will not hallucinate random outputs for most of the training data. Therefore, the set of predictions $\hat{\boldsymbol{y}}$ would still provide useful information for the optimizer, even though it is noisy. This is exactly what happened for many of our experiments. For instance, in Section 4.2 Figure 4(b; both mid and right) we can see a small outlier (a dent) in the plot of the quadratic function, which is due to such hallucination. Nevertheless, VML is still able to learn a very good model.

Hallucination in $f_{\text{opt}}(\cdot)$ is when the optimizer does not produce a sensible model parameter for the current step of training. From our experiment section, we can indeed see that many of our case studies have a non-monotonic training loss, some of the fluctuation can be explained by hallucination (see failure cases in Appendix C for example). However, since hallucination is a low probability event, an unsatisfied model

parameter will most likely be corrected in the next step of optimization. Therefore, in most runs, VML eventually learns a very good model. For cases where it cannot correct itself from the hallucination, they will be identified as failure cases by simply checking the learn model, the training loss, or the test set performance. As discussed in Appendix C, by adding more detailed instruction to the optimizer prompt template and switching to a more powerful LLM, these failure cases (a superset of hallucination) will occur less frequently.

Across our experiments, we did not witness any empirical evidence that hallucination is affecting the reliability of VML. The core of VML is that a model is characterized by the text prompt of an LLM. Whether the model parameter is good depends on its downstream performance (*e.g.*, the training accuracy). Therefore, if a model parameter works well in the downstream task, it is highly unlikely that the model parameter is based on hallucination, because the hallucinated text is unlikely to demonstrate consistently good performance for all the data points. If the learned well-performing model parameter seems unexpected, it is more likely that the LLM-based optimizer discovered new knowledge from the training data than have hallucinated.

More importantly, hallucination in LLMs does not limit the usefulness and significance of using LLMs to solve numerical tasks (such as solving math problems). Therefore, we don't view hallucination as a problem for VML, but rather an opportunity for future improvement. In reality, we observe that hallucination does not limit the development of utilizing LLMs to solve math problems, which requires the capability of LLMs in understanding numbers and mathematical functions. The math task even requires more precise reasoning from LLMs. However, LLMs has still achieved tremendous progress in math reasoning, despite the potential of hallucination. For example, DeepMind's AlphaProof[3] (based on Gemini) recently reached IMO silver level performance. One can also observe the progress of LLMs on the MATH dataset from around 15% [52] to 84% [62] (The MATH dataset has competition-level mathematics problems with many numerical computations). The inherited problem of hallucination does not seem to invalidate the field of AI for math. Therefore, it is hard to be certain, in particular without empirical evidence, that hallucination will be a critical limitation for VML.

We agree that current LLMs have many problems (*e.g.*, hallucination, finite context-window). All these shortcomings of LLMs are being actively studied today. We believe VML is actually an orthogonal & independent contribution to these existing LLM research topics. VML studies how to enable interpretable learning using natural language. As LLM gets more powerful and more faithful, VML will also become more useful.

We also want to highlight that the goal of this work is not to propose a method to replace numerical machine learning, nor to claim superiority of VML over numerical machine learning. We are simply sharing this new possibility of doing learning in the LLMs era, and showing evidence that it can indeed learn for many tasks. VML does have many features which the numerical machine learning does not have, such as interpretability and adding prior with natural language, but it is also not scalable at the moment.

L Complete Training Template at Initialization

L.1 Linear Regression

Text prompt template for the learner

You are the model. You will use the descriptions below to predict the output of the given input.

** Pattern Descriptions: **

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.

** Input: **

[$Data]

Please give your output strictly in the following format:

```
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
```

Please ONLY reply according to this format, don't give me any other words.

Text prompt template for the optimizer

You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.

** Inputs (a batch of i.i.d. data): **

[[$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data]]

** Current Pattern Descriptions: **

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.

** The model outputs: **

[[$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction]]

** The target outputs: **

[[$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth]]

If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Please think step by step and give your outputs strictly in the following format:

```
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself;]

New Pattern Descriptions:

[put your new descriptions here; MUST be specific and concrete;]
```

Please ONLY reply according to this format, don't give me any other words.

Figure 27: Prompt templates of VML for the learner and optimizer for the linear regression (Llama-3-70B without prior).

L.2 Polynomial Regression

Text prompt template for the learner

You are the model. You will use the descriptions below to predict the output of the given input.

** Pattern Descriptions: **

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.

** Input: **

[$Data]

Please give your output strictly in the following format:
```
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
```
Please ONLY reply according to this format, don't give me any other words.

Text prompt template for the optimizer

You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.

** Inputs (a batch of i.i.d. data): **

[[$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data]]

** Current Pattern Descriptions: **

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.

** The model outputs: **

[[$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction]]

** The target outputs: **

[[$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth]]

If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:

```
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
```
Please ONLY reply according to this format, don't give me any other words.

Figure 28: Prompt templates of VML for the learner and optimizer for the polynomial regression (Llama-3-70B without prior).

L.3 Sinusoidal Regression

Text prompt template for the learner

You are the model. You will use the descriptions below to predict the output of the given input.

** Pattern Descriptions: **

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. *It looks like the data is generated by a periodic function.*

** Input: **

[$Data]

Please give your output strictly in the following format:

```
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
```

Please ONLY reply according to this format, don't give me any other words.

Text prompt template for the optimizer

You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.

** Inputs (a batch of i.i.d. data): **

[[$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data]]

** Current Pattern Descriptions: **

You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function.

** The model outputs: **

[[$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction]]

** The target outputs: **

[[$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth]]

If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:

```
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
```

Please ONLY reply according to this format, don't give me any other words.

Figure 29: Prompt templates of VML for the learner and optimizer for the sinusoidal regression (GPT-4o with *prior*).

L.4 Two Blobs Classification

Text prompt template for the learner

You are the model.

** Model Descriptions: **

You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0.

** Input: **

[$Data]

Please give your output strictly in the following format:
```
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
```
Please ONLY reply according to this format, don't give me any other words.

Text prompt template for the optimizer

You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.

** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): **

[[$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data]]

** Current Model Descriptions: **

You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, output [0.0, 1.0]. If x < 0, if y < 8.5, output [0.0, 1.0], otherwise output [1.0, 0.0].

** The model predictions ([class 1 prob. class 2 prob.]): **

[[$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction]]

** The targets ([class 1 prob. class 2 prob.]): **

[[$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth]]

Please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:

```
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
```

Please ONLY reply according to this format, don't give me any other words.

Figure 30: Prompt templates of VML for the learner and optimizer for the two blobs classification (Llama-3-70B without prior).

L.5 Two Circles Classification

Text prompt template for the learner

You are the model.

** Model Descriptions: **

You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set {0, 1}. *The decision boundary is a circle.*

** Input: **

[$Data]

Please give your output strictly in the following format:

```
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
```

Please ONLY reply according to this format, don't give me any other words.

Text prompt template for the optimizer

You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.

** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): **

[[$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data]]

** Current Model Descriptions: **

You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set {0, 1}. *The decision boundary is a circle.*

** The model predictions: **

[[$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction]]

** The targets: **

[[$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth]]

If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:

```
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
```

Please ONLY reply according to this format, don't give me any other words.

Figure 31: Prompt templates of VML for the learner and optimizer for the two circles classification (Llama-3-70B with prior).

L.6 Text classification

Text prompt template for the learner

You are the model.

** Model Descriptions: **

You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set {0, 1}.

** Input: **

[$Data]

Please give your output strictly in the following format:

```
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
```

Please ONLY reply according to this format, don't give me any other words.

Text prompt template for the optimizer

You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.

** Inputs (a batch of i.i.d. text): **

[[$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data] [$Data]]

** Current Model Descriptions: **

You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set {0, 1}.

** The model predictions: **

[[$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction] [$Prediction]]

** The targets: **

[[$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth] [$GroundTruth]]

If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Please think step by step and give your outputs strictly in the following format:

```
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here]
```

Please ONLY reply according to this format, don't give me any other words.

Figure 32: Prompt templates of VML for the learner and optimizer for the text classification (Llama-3-70B without prior).

M Detailed Training History

M.1 Linear Regression (Llama-3-70B without prior)

Training History: Linear Regression – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.
**** Input: ****
[1.26]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: Since there is no specific pattern or function provided, I will make an assumption that the output is the same as the input.
Output: [1.89]
———— **Forward Pass Output** ————
Explanations: Not enough information is provided to make a meaningful prediction. Without a clear pattern or relationship, I will assume a simple linear model.
Output: [3.70]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: Since there's no specific pattern or formula provided, I'll make an assumption based on the given input. The input [0.26] seems to be a small positive value, so I'll predict the output to be slightly larger, but still a small positive value.
Output: [0.31]
———— **Overall Loss** ————
36.34138
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[1.26] [1.8] [0.99] [1.86] [0.36] [0.36] [0.97] [1.85] [0.26] [1.89]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.
**** The model outputs: ****
[[1.27] [3.6] [0.98] [1.86] [0.42] [0.5] [0.97] [3.7] [0.31] [1.89]]
**** The target outputs: ****
[[7.75] [9.43] [6.95] [9.55] [4.88] [5.23] [6.96] [9.62] [4.75] [9.62]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model outputs are not close to the target outputs, indicating that the current pattern descriptions are not effective. Upon examining the inputs and target outputs, I notice that the target outputs are generally larger than the inputs, suggesting a possible linear relationship with a positive slope. However, the current model outputs are mostly similar to the inputs, indicating that the current pattern descriptions do not capture this relationship. To optimize the pattern descriptions, I will introduce a scaling factor to capture the linear relationship between the inputs and target outputs. I will also add a small bias term to account for any constant shift in the target outputs.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.5. The new pattern descriptions are: $y = 5.5x + 2.5$ where y is the predicted output and x is the input.
———— **Updated Theta** ————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.5. The new pattern descriptions are:
$y = 5.5x + 2.5$
where y is the predicted output and x is the input.

Training History: Linear Regression – Step 2

Epoch 0 - Step 2
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.5. The new pattern descriptions are:
y = 5.5x + 2.5
where y is the predicted output and x is the input.
**** Input: ****
[0.86]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Forward Pass Output** —————
Explanations:
Given input x = 0.28, we can plug it into the equation y = 5.5x + 2.5 to get the output y.
y = 5.5(0.28) + 2.5
y = 1.54 + 2.5
y = 4.04
Output: [4.04]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
The given linear regression model is y = 5.5x + 2.5. We need to find the output y for the given input x = 1.59. Plugging in the value of x into the equation, we get y = 5.5(1.59) + 2.5.
First, let's calculate 5.5(1.59) = 8.745. Now, add 2.5 to get y = 8.745 + 2.5 = 11.245. Rounded to two decimal points, the output is 11.25.
Output: [11.25]
————— **Overall Loss** —————
2.8677400000000004
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[0.86] [0.02] [1.1] [1.59] [1.2] [0.28] [1.89] [1.29] [0.79] [0.01]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.5. The new pattern descriptions are:
y = 5.5x + 2.5
where y is the predicted output and x is the input.
**** The model outputs: ****
[[7.21] [2.61] [8.55] [11.25] [9.1] [4.04] [12.92] [9.59] [6.85] [2.56]]
**** The target outputs: ****
[[6.56] [3.98] [7.29] [8.6] [7.7] [4.77] [9.67] [8.03] [6.43] [4.01]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Optimization Output** —————
Reasoning:
The current model outputs are not close to the target outputs, indicating that the current pattern descriptions are not effective. Upon examining the inputs, target outputs, and model outputs, I notice that the model outputs are generally larger than the target outputs, suggesting that the scaling factor of 5.5 is too large. I will reduce the scaling factor to bring the model outputs closer to the target outputs. Additionally, I will adjust the bias term to better capture the overall shift in the target outputs. To optimize the pattern descriptions, I will calculate the average difference between the target outputs and model outputs to determine the adjustment needed for the bias term. The average difference is approximately -1.45. I will subtract this value from the current bias term of 2.5 to get a new bias term of 1.05.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 4.2 and a bias term of 1.05. The new pattern descriptions are:
y = 4.2x + 1.05
where y is the predicted output and x is the input.
————— **Updated Theta** —————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 4.2 and a bias term of 1.05. The new pattern descriptions are:
y = 4.2x + 1.05
where y is the predicted output and x is the input.

Training History: Linear Regression – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 4.2 and a bias term of 1.05. The new pattern descriptions are:
y = 4.2x + 1.05
where y is the predicted output and x is the input.
**** Input: ****
[0.49]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output: [Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Using the given linear regression model, substitute x with the input 0.05.
y = 4.2x + 1.05
y = 4.2(0.05) + 1.05
y = 0.21 + 1.05
y = 1.26
Output: [1.26]

Many outputs for the rest data points ...

———— **Overall Loss** ————
4.176069999999999
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[0.49] [0.74] [0.13] [0.05] [0.84] [1.21] [1.28] [1.77] [0.99] [0.77]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 4.2 and a bias term of 1.05. The new pattern descriptions are:
y = 4.2x + 1.05
where y is the predicted output and x is the input.
**** The model outputs: ****
[[3.11] [4.16] [1.6] [1.26] [4.58] [6.13] [6.43] [8.52] [5.21] [4.28]]
**** The target outputs: ****
[[5.59] [6.22] [4.31] [4.06] [6.36] [7.75] [7.87] [9.17] [6.97] [6.47]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model outputs are not close to the target outputs, indicating that the current pattern descriptions are not effective. Upon examining the inputs, target outputs, and model outputs, I notice that the model outputs are generally smaller than the target outputs, suggesting that the scaling factor of 4.2 is too small. I will increase the scaling factor to bring the model outputs closer to the target outputs. Additionally, I will adjust the bias term to better capture the overall shift in the target outputs. To optimize the pattern descriptions, I will calculate the average difference between the target outputs and model outputs to determine the adjustment needed for the bias term. The average difference is approximately 1.43. I will add this value to the current bias term of 1.05 to get a new bias term of 2.48. I will also analyze the relationship between the inputs and target outputs to determine the new scaling factor. Upon examining the data, I notice that the target outputs increase by approximately 5.5 units for every 1 unit increase in the input. I will use this relationship to set the new scaling factor to 5.5.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.48. The new pattern descriptions are:
y = 5.5x + 2.48
where y is the predicted output and x is the input.
———— **Updated Theta** ————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.48. The new pattern descriptions are:
y = 5.5x + 2.48
where y is the predicted output and x is the input.

Training History: Linear Regression – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.48. The new pattern descriptions are:
y = 5.5x + 2.48
where y is the predicted output and x is the input.
**** Input: ****
[0.08]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given the input x = 1, I will use the linear regression model y = 5.5x + 2.48 to predict the output y.
Substituting x = 1 into the equation, I get y = 5.5(1) + 2.48 = 7.98.
Output: [7.98]

Many outputs for the rest data points ...

———— **Overall Loss** ————
2.6112699999999993
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[0.08] [0.44] [0.11] [0.84] [1.48] [0.02] [1.] [1.39] [1.7] [1.38]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.5 and a bias term of 2.48. The new pattern descriptions are:
y = 5.5x + 2.48
where y is the predicted output and x is the input.
**** The model outputs: ****
[[2.92] [4.9] [3.09] [7.12] [10.62] [2.59] [7.98] [10.1] [11.93] [10.07]]
**** The target outputs: ****
[[4.27] [5.19] [4.24] [6.55] [8.61] [3.89] [6.96] [8.15] [9.17] [7.98]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model outputs are not close to the target outputs, indicating that the current pattern descriptions are not effective. Upon examining the inputs, target outputs, and model outputs, I notice that the model outputs are generally smaller than the target outputs, suggesting that the bias term of 2.48 is too small. I will increase the bias term to bring the model outputs closer to the target outputs. Additionally, I will adjust the scaling factor to better capture the relationship between the inputs and target outputs. To optimize the pattern descriptions, I will calculate the average difference between the target outputs and model outputs to determine the adjustment needed for the bias term. The average difference is approximately 1.23. I will add this value to the current bias term of 2.48 to get a new bias term of 3.71. I will also analyze the relationship between the inputs and target outputs to determine the new scaling factor. Upon examining the data, I notice that the target outputs increase by approximately 5.8 units for every 1 unit increase in the input. However, the current scaling factor of 5.5 is close to this value, so I will only make a small adjustment to the scaling factor. I will increase the scaling factor to 5.7 to better capture the relationship between the inputs and target outputs.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.7 and a bias term of 3.71. The new pattern descriptions are:
y = 5.7x + 3.71
where y is the predicted output and x is the input.
———— **Updated Theta** ————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.7 and a bias term of 3.71. The new pattern descriptions are:
y = 5.7x + 3.71
where y is the predicted output and x is the input.

Training History: Linear Regression – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.7 and a bias term of 3.71. The new pattern descriptions are:
y = 5.7x + 3.71
where y is the predicted output and x is the input.
**** Input: ****
[1.59]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given the input x = 1.39, we can plug it into the equation y = 5.7x + 3.71 to get the output y.
y = 5.7(1.39) + 3.71
y = 7.813 + 3.71
y = 11.523
Output: [11.52]

Many outputs for the rest data points ...

———— **Overall Loss** ————
8.08032
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[1.59] [0.23] [1.72] [0.98] [1.39] [1.7] [1.54] [0.49] [0.27] [0.93]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 5.7 and a bias term of 3.71. The new pattern descriptions are:
y = 5.7x + 3.71
where y is the predicted output and x is the input.
**** The model outputs: ****
[[12.79] [5.02] [13.51] [9.3] [11.52] [13.4] [11.42] [6.49] [5.25] [8.99]]
**** The target outputs: ****
[[9.02] [4.64] [9.2] [7.02] [8.33] [9.14] [8.55] [5.47] [4.77] [6.79]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model outputs are not close to the target outputs, indicating that the current pattern descriptions are not effective. Upon examining the inputs, target outputs, and model outputs, I notice that the model outputs are generally larger than the target outputs, suggesting that the scaling factor of 5.7 is too large. I will decrease the scaling factor to bring the model outputs closer to the target outputs. Additionally, I will adjust the bias term to better capture the relationship between the inputs and target outputs. To optimize the pattern descriptions, I will calculate the average difference between the target outputs and model outputs to determine the adjustment needed for the bias term. The average difference is approximately -1.44. I will subtract this value from the current bias term of 3.71 to get a new bias term of 2.27. I will also analyze the relationship between the inputs and target outputs to determine the new scaling factor. Upon examining the data, I notice that the target outputs increase by approximately 4.2 units for every 1 unit increase in the input. The current scaling factor of 5.7 is larger than this value, so I will decrease the scaling factor to 4.4 to better capture the relationship between the inputs and target outputs.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 4.4 and a bias term of 2.27. The new pattern descriptions are:
y = 4.4x + 2.27
where y is the predicted output and x is the input.
———— **Updated Theta** ————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 4.4 and a bias term of 2.27. The new pattern descriptions are:
y = 4.4x + 2.27
where y is the predicted output and x is the input.

Training History: Linear Regression – Step 19

Epoch 1 - Step 9
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.26 and a bias term of 3.45. The new pattern descriptions are:
y = 3.26x + 3.45
where y is the predicted output and x is the input.
**** Input: ****
[1.26]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Forward Pass Output** —————
Explanations:
Given the linear regression model y = 3.26x + 3.45, we can plug in the input x = 1.26.
y = 3.26(1.26) + 3.45 = 4.08 + 3.45 = 7.53.
Output: [7.53]

Many outputs for the rest data points ...

————— **Overall Loss** —————
0.11138999999999992
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[1.26] [1.66] [0.32] [0.83] [1.3] [0.24] [1.87] [1.15] [1.2] [1.26]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.26 and a bias term of 3.45. The new pattern descriptions are:
y = 3.26x + 3.45
where y is the predicted output and x is the input.
**** The model outputs: ****
[[7.53] [8.88] [4.49] [6.17] [7.71] [4.23] [9.58] [7.19] [7.36] [7.54]]
**** The target outputs: ****
[[7.87] [8.93] [4.93] [6.36] [8.18] [4.78] [9.37] [7.39] [7.71] [7.73]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words. ************ Shortening the state from 5 to 3 ************
————— **Optimization Output** —————
Reasoning:
The current model outputs are generally close to the target outputs, but there are some deviations. Upon examining the inputs, target outputs, and model outputs, I notice that the model outputs are still underestimating the target outputs for some inputs, and overestimating for others. This suggests that the bias term of 3.45 might not be optimal. To optimize the pattern descriptions, I will calculate the average difference between the model outputs and target outputs. The average difference is approximately 0.09. I will add this value to the current bias term of 3.45 to get a new bias term of 3.54. I will also analyze the relationship between the inputs and target outputs to determine if the scaling factor of 3.26 needs to be adjusted. Upon examining the data, I notice that the target outputs are still increasing with the inputs, and the rate of increase is slightly faster than the current scaling factor of 3.26. This suggests that the scaling factor of 3.26 might be slightly too small. I will increase the scaling factor by 0.01 to get a new scaling factor of 3.27.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.27 and a bias term of 3.54. The new pattern descriptions are:
y = 3.27x + 3.54
where y is the predicted output and x is the input.
————— **Updated Theta** —————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.27 and a bias term of 3.54. The new pattern descriptions are:
y = 3.27x + 3.54
where y is the predicted output and x is the input.

Training History: Linear Regression – Step 20

Epoch 1 - Step 10
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.27 and a bias term of 3.54. The new pattern descriptions are:
y = 3.27x + 3.54
where y is the predicted output and x is the input.
**** Input: ****
[0.71]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Using the given linear regression model, y = 3.27x + 3.54, and the input x = 1, we can calculate the output y as follows:
y = 3.27(1) + 3.54
y = 3.27 + 3.54
y = 6.81
Output: 6.81

Many outputs for the rest data points ...

———— **Overall Loss** ————
0.08741999999999994
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[0.71] [0.6] [0.51] [1.34] [0.92] [0.16] [1.12] [0.14] [1.67] [1.]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.27 and a bias term of 3.54. The new pattern descriptions are:
y = 3.27x + 3.54
where y is the predicted output and x is the input.
**** The model outputs: ****
[[5.86] [5.5] [5.22] [7.91] [6.57] [4.07] [7.21] [3.92] [9.] [6.81]]
**** The target outputs: ****
[[6.05] [5.8] [5.48] [8.1] [6.56] [4.63] [7.37] [4.43] [8.85] [6.96]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model outputs are generally close to the target outputs, but there are some deviations. Upon examining the inputs, target outputs, and model outputs, I notice that the model outputs are still underestimating the target outputs for some inputs, and overestimating for others. This suggests that the bias term of 3.54 might not be optimal. To optimize the pattern descriptions, I will calculate the average difference between the model outputs and target outputs. The average difference is approximately 0.15. I will add this value to the current bias term of 3.54 to get a new bias term of 3.69. I will also analyze the relationship between the inputs and target outputs to determine if the scaling factor of 3.27 needs to be adjusted. Upon examining the data, I notice that the target outputs are still increasing with the inputs, and the rate of increase is slightly faster than the current scaling factor of 3.27. This suggests that the scaling factor of 3.27 might be slightly too small. I will increase the scaling factor by 0.02 to get a new scaling factor of 3.29.
New Pattern Descriptions:
The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.29 and a bias term of 3.69. The new pattern descriptions are:
y = 3.29x + 3.69
where y is the predicted output and x is the input.
———— **Updated Theta** ————
The new pattern descriptions will be based on a linear regression model with a scaling factor of 3.29 and a bias term of 3.69. The new pattern descriptions are:
y = 3.29x + 3.69
where y is the predicted output and x is the input.

M.2 Polynomial Regression (Llama-3-70B without prior)

Training History: Polynomial Regression – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.
**** Input: ****
[-2.4]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations: Since there is no specific pattern or relationship provided, I will make an assumption that the output is a constant value. In this case, I will assume the output is 0.00.
Output: [0.00]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Since there is no specific pattern or function provided, I will make an assumption that the input and output are linearly related with a constant difference of 1.2. Based on this assumption, I will add 1.2 to the input to get the output.
Output: [0.50]
———— **Overall Loss** ————
63.153999999999996
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-2.4] [0.6] [0.9] [-1.8] [-0.7] [-0.9] [-1.2] [-0.] [-0.2] [0.3]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers.
**** The model outputs: ****
[[-2.4] [0.] [0.] [0.] [0.5] [0.] [-1.2] [0.] [-0.2] [0.5]]
**** The target outputs: ****
[[17.8] [3.7] [6.6] [10.] [3.6] [2.5] [5.4] [1.1] [0.9] [2.6]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the ’New Pattern Descriptions’. The model uses the ’New Pattern Descriptions’ should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don’t have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
After examining the model outputs and target outputs, it’s clear that the model is not performing well. The model outputs are mostly around 0, while the target outputs have a larger range of values. This suggests that the model is not capturing the underlying pattern in the data. Upon closer inspection, I notice that the inputs and target outputs seem to have a positive correlation. This means that as the input increases, the target output also tends to increase. To improve the model, I will introduce a new pattern description that captures this positive correlation. I will use a simple linear relationship between the input and output, with a slope and intercept that can be learned from the data.
New Pattern Descriptions:
The new pattern description is: output = 2.5 * input + 1.5.
———— **Updated Theta** ————
The new pattern description is: output = 2.5 * input + 1.5.

Training History: Polynomial Regression – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 2.5 * input + 1.5.
**** Input: ****
[-2.4]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The output is calculated by multiplying the input by 2.5 and adding 1.5. In this case, the input is 1, so the output is 2.5 * 1 + 1.5 = 4.
Output: [4.00]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
To find the output, we need to apply the given pattern description: output = 2.5 * input + 1.5.
In this case, the input is -1.2, so we plug it into the equation: output = 2.5 * (-1.2) + 1.5.
First, we multiply 2.5 and -1.2: 2.5 * (-1.2) = -3.0.
Then, we add 1.5: -3.0 + 1.5 = -1.5.
Output: [-1.50]
———— **Overall Loss** ————
257.99
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-2.4] [-2.9] [-2.1] [-1.8] [-1.2] [-0.6] [-2.2] [1.] [-1.9] [1.]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 2.5 * input + 1.5.
**** The model outputs: ****
[[-4.5] [-5.8] [-3.8] [-3.] [-1.5] [0.] [-4.] [4.] [-3.2] [4.]]
**** The target outputs: ****
[[19.5] [26.1] [12.4] [10.8] [4.5] [0.8] [13.8] [7.4] [9.7] [5.4]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
After examining the model outputs and target outputs, it's clear that the model is not performing well. The model outputs are not close to the target outputs, and the pattern description output = 2.5 * input + 1.5 seems to be oversimplifying the relationship between the input and output. Upon closer inspection, I notice that the target outputs are generally larger than the model outputs. This suggests that the model is underestimating the target outputs. I also notice that the target outputs seem to have a non-linear relationship with the inputs, with larger inputs corresponding to larger outputs, but not in a linear way. To improve the model, I will introduce a new pattern description that captures this non-linear relationship. I will use a quadratic relationship between the input and output, with coefficients that can be learned from the data. This will allow the model to better fit the curved relationship between the inputs and target outputs.
New Pattern Descriptions:
The new pattern description is: output = 2.2 * input^2 + 1.8 * input + 0.6.
———— **Updated Theta** ————
The new pattern description is: output = 2.2 * input^2 + 1.8 * input + 0.6.

Training History: Polynomial Regression – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 2.2 * input^2 + 1.8 * input + 0.6.
**** Input: ****
[-1.]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given input = -1.9, we need to find output using the formula: output = 2.2 * input^2 + 1.8 * input + 0.6.
First, calculate input^2 = (-1.9)^2 = 3.61.
Now, calculate output = 2.2 * 3.61 + 1.8 * (-1.9) + 0.6 = -2.42.
Output: [-2.42]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Given the input 0.8, we need to plug it into the equation output = 2.2 * input^2 + 1.8 * input + 0.6.
First, we square the input: 0.8^2 = 0.64.
Then, we multiply 0.64 by 2.2: 2.2 * 0.64 = 1.408.
Next, we multiply 0.8 by 1.8: 1.8 * 0.8 = 1.44.
Now, we add 1.408 and 1.44: 1.408 + 1.44 = 2.848.
Finally, we add 0.6 to 2.848: 2.848 + 0.6 = 3.448.
Output: [3.45]
———— **Overall Loss** ————
44.34
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-1.] [-2.7] [0.6] [-1.9] [-2.6] [-1.8] [-2.1] [0.8] [-0.8] [-2.1]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 2.2 * input^2 + 1.8 * input + 0.6.
**** The model outputs: ****
[[1.] [12.8] [2.5] [-2.4] [10.8] [4.5] [6.5] [3.4] [0.6] [6.5]]
**** The target outputs: ****
[[4.2] [21.2] [2.4] [11.7] [18.1] [8.3] [12.2] [6.8] [1.] [13.7]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
After examining the model outputs and target outputs, it's clear that the model is not performing well. The model outputs are not close to the target outputs, and the pattern description output = 2.2 * input^2 + 1.8 * input + 0.6 seems to be undershooting the target outputs. Upon closer inspection, I notice that the target outputs are generally larger than the model outputs, especially for larger inputs. This suggests that the model is underestimating the target outputs, especially for larger inputs. I also notice that the target outputs seem to have a stronger quadratic relationship with the inputs, with larger inputs corresponding to even larger outputs. To improve the model, I will introduce a new pattern description that captures this stronger quadratic relationship. I will increase the coefficient of the quadratic term to capture the stronger curvature of the relationship. I will also adjust the linear term to better fit the data.
New Pattern Descriptions:
The new pattern description is: output = 2.8 * input^2 + 1.2 * input + 0.4.
———— **Updated Theta** ————
The new pattern description is: output = 2.8 * input^2 + 1.2 * input + 0.4.

Training History: Polynomial Regression – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 2.8 * input^2 + 1.2 * input + 0.4.
**** Input: ****
[0.2]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given the pattern description: output = 2.8 * input^2 + 1.2 * input + 0.4.
Input is 0.4. So, output = 2.8 * (0.4)^2 + 1.2 * 0.4 + 0.4 = 0.448 + 0.48 + 0.4 = 1.328.
Output: [1.33]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Given the input -1.7, we need to find the output using the formula output = 2.8 * input^2 + 1.2 * input + 0.4.
First, we calculate the square of the input: (-1.7)^2 = 2.89.
Then, we multiply it by 2.8: 2.89 * 2.8 = 8.092.
Next, we multiply the input by 1.2: -1.7 * 1.2 = -2.04.
Now, we add 8.092 and -2.04: 8.092 - 2.04 = 6.052.
Finally, we add 0.4: 6.052 + 0.4 = 6.452.
Output: [6.45]
———— **Overall Loss** ————
8.585
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[0.2] [-1.7] [-2.8] [-1.1] [0.4] [0.8] [-0.1] [-0.1] [0.8] [-0.7]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 2.8 * input^2 + 1.2 * input + 0.4.
**** The model outputs: ****
[[0.8] [6.4] [19.] [2.5] [1.3] [3.2] [0.3] [0.3] [3.2] [-0.5]]
**** The target outputs: ****
[[3.] [8.6] [25.6] [6.5] [1.2] [5.5] [0.7] [0.8] [5.1] [2.2]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
After examining the model outputs and target outputs, it's clear that the model is not performing well. The model outputs are not close to the target outputs, and the pattern description output = 2.8 * input^2 + 1.2 * input + 0.4 seems to be undershooting the target outputs, especially for larger inputs. Upon closer inspection, I notice that the target outputs are generally larger than the model outputs, especially for larger inputs. This suggests that the model is underestimating the target outputs, especially for larger inputs. I also notice that the target outputs seem to have an even stronger quadratic relationship with the inputs, with larger inputs corresponding to even larger outputs. To improve the model, I will introduce a new pattern description that captures this even stronger quadratic relationship. I will increase the coefficient of the quadratic term to capture the stronger curvature of the relationship. I will also adjust the linear term to better fit the data.
New Pattern Descriptions:
The new pattern description is: output = 3.2 * input^2 + 0.8 * input + 0.6.
———— **Updated Theta** ————
The new pattern description is: output = 3.2 * input^2 + 0.8 * input + 0.6.

Training History: Polynomial Regression – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 3.2 * input^2 + 0.8 * input + 0.6.
**** Input: ****
[-2.3]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given the pattern description output = 3.2 * input^2 + 0.8 * input + 0.6, and the input is -2.3,
first, we calculate input^2 which is (-2.3)^2 = 5.29.
Then, we plug in the values into the equation: output = 3.2 * 5.29 + 0.8 * -2.3 + 0.6 = 16.928 - 1.84 + 0.6 = 15.708.
Output: [15.71]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Given the pattern description output = 3.2 * input^2 + 0.8 * input + 0.6, I will use this equation to find the output for the given input -2.4.
First, I will calculate input^2: (-2.4)^2 = 5.76.
Next, I will calculate 3.2 * input^2: 3.2 * 5.76 = 18.43.
Then, I will calculate 0.8 * input: 0.8 * -2.4 = -1.92.
Now, I will add the results of the previous steps: 18.43 - 1.92 + 0.6 = 17.11.
Output: [17.11]
———— **Overall Loss** ————
2.366000000000002
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-2.3] [0.3] [-2.4] [0.5] [-2.3] [-1.4] [-1.7] [-3.] [-2.3] [-0.5]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 3.2 * input^2 + 0.8 * input + 0.6.
**** The model outputs: ****
[[15.] [1.1] [17.1] [1.8] [15.7] [5.8] [8.5] [27.] [15.7] [1.]]
**** The target outputs: ****
[[14.8] [2.4] [17.1] [2.9] [17.3] [6.3] [8.7] [22.9] [16.6] [1.5]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
After examining the model outputs and target outputs, it's clear that the model is not performing well, especially for larger inputs. The model outputs are generally higher than the target outputs, indicating that the model is overshooting the target outputs. Upon closer inspection, I notice that the target outputs seem to have a stronger quadratic relationship with the inputs, especially for larger inputs. The current pattern description output = 3.2 * input^2 + 0.8 * input + 0.6 seems to be too steep, resulting in overshooting. To improve the model, I will adjust the coefficient of the quadratic term to reduce the steepness of the relationship. I will also adjust the linear term to better fit the data. Additionally, I will adjust the constant term to better fit the data.
New Pattern Descriptions:
The new pattern description is: output = 2.9 * input^2 + 0.5 * input + 0.4.
———— **Updated Theta** ————
The new pattern description is: output = 2.9 * input^2 + 0.5 * input + 0.4.

Training History: Polynomial Regression – Step 19

Epoch 1 - Step 9
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 3.2 * input^2 + 1.8 * input + 1.7.
**** Input: ****
[-1.5]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Forward Pass Output** —————
Explanations:
The given input is 0.
Using the pattern description output = 3.2 * input^2 + 1.8 * input + 1.7,
we get output = 3.2 * 0^2 + 1.8 * 0 + 1.7 = 1.7.
Output: [1.70]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
The given pattern description is output = 3.2 * input^2 + 1.8 * input + 1.7.
We are given an input of 0.3.
First, we square the input to get 0.3^2 = 0.09.
Then, we multiply 0.09 by 3.2 to get 3.2 * 0.09 = 0.288.
Next, we multiply the input by 1.8 to get 1.8 * 0.3 = 0.54.
Finally, we add 1.7 to the sum of the previous results to get 0.288 + 0.54 + 1.7 = 2.528.
Output: [2.53]
————— **Overall Loss** —————
1.1499999999999995
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-1.5] [-1.8] [-0.8] [0.] [0.3] [-2.6] [0.9] [-0.1] [0.2] [-0.5]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 3.2 * input^2 + 1.8 * input + 1.7.
**** The model outputs: ****
[[6.2] [8.8] [2.3] [1.7] [2.5] [18.6] [5.9] [1.6] [2.2] [1.6]]
**** The target outputs: ****
[[7.8] [10.9] [2.6] [1.5] [3.9] [19.5] [4.8] [1.1] [1.8] [1.5]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Optimization Output** —————
Reasoning:
After examining the model outputs and target outputs, I notice that the model is still not capturing the curvature of the data well, especially for inputs with large absolute values. The current pattern description output = 3.2 * input^2 + 1.8 * input + 1.7 seems to be underestimating the outputs for inputs with large absolute values. To improve the model, I will increase the coefficient of the quadratic term to better fit the data, and adjust the linear term to better capture the trend. I will consider the previous optimization step, where I decreased the coefficient of the quadratic term from 3.4 to 3.2. This time, I will increase it to 3.3 to better capture the curvature. Upon examining the data, I notice that the model is performing poorly for inputs with large absolute values. I will increase the coefficient of the quadratic term to 3.3 to better capture the curvature of the data for these inputs. I will also adjust the linear term to 1.9 to better capture the trend. Finally, I will adjust the constant term to 1.6 to better fit the data.
New Pattern Descriptions:
The new pattern description is: output = 3.3 * input^2 + 1.9 * input + 1.6.
********** Updated Theta **********
The new pattern description is: output = 3.3 * input^2 + 1.9 * input + 1.6.

Training History: Polynomial Regression – Step 20

Epoch 1 - Step 10
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 3.3 * input^2 + 1.9 * input + 1.6.
**** Input: ****
[-0.1]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given the input -3, we need to calculate the output using the formula: output = 3.3 * input^2 + 1.9 * input + 1.6.
First, calculate input^2: (-3)^2 = 9.
Now, substitute the values: output = 3.3 * 9 + 1.9 * (-3) + 1.6.
output = 29.7 - 5.7 + 1.6.
output = 25.6.
Output: [25.60]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Given the input 0.9, we need to apply the formula output = 3.3 * input^2 + 1.9 * input + 1.6.
First, we calculate the square of the input: 0.9^2 = 0.81.
Then, we multiply 0.81 by 3.3: 0.81 * 3.3 = 2.673.
Next, we multiply 0.9 by 1.9: 0.9 * 1.9 = 1.71.
Now, we add 2.673 and 1.71: 2.673 + 1.71 = 4.383.
Finally, we add 1.6 to 4.383: 4.383 + 1.6 = 5.983.
Output: [5.98]
———— **Overall Loss** ————
1.5080000000000002
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-0.1] [0.6] [0.7] [-3.] [0.9] [-1.9] [-2.6] [0.8] [-0.3] [0.2]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. The new pattern description is: output = 3.3 * input^2 + 1.9 * input + 1.6.
**** The model outputs: ****
[[1.4] [3.9] [4.6] [25.6] [6.] [6.3] [18.9] [5.2] [1.3] [2.1]]
**** The target outputs: ****
[[1.1] [4.] [3.6] [26.7] [4.1] [8.6] [20.1] [5.7] [2.6] [2.8]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
After examining the model outputs and target outputs, I notice that the model is still not capturing the curvature of the data well, especially for inputs with large absolute values. The current pattern description output = 3.3 * input^2 + 1.9 * input + 1.6 seems to be underestimating the outputs for inputs with large absolute negative values. To improve the model, I will increase the coefficient of the quadratic term to better fit the data, and adjust the linear term to better capture the trend. I will also adjust the constant term to better fit the data. Upon examining the data, I notice that the model is performing poorly for inputs with large absolute negative values. I will increase the coefficient of the quadratic term to 3.4 to better capture the curvature of the data for these inputs. I will also adjust the linear term to 2.1 to better capture the trend. Finally, I will adjust the constant term to 1.5 to better fit the data.
New Pattern Descriptions:
The new pattern description is: output = 3.4 * input^2 + 2.1 * input + 1.5.
************ Updated Theta ************
The new pattern description is: output = 3.4 * input^2 + 2.1 * input + 1.5.

M.3 Sinusoidal Regression (GPT-4o with prior)

Training History: Sinusoidal Regression – Step 1

Epoch 0 - Step 1
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function.
** Input: **
[-1.8]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Forward Pass Output** —————
Explanations:
The data is generated by a periodic function. Since the function is periodic, it might resemble common periodic functions such as sine or cosine. For the input -0.8, we will assume the function could be related to the sine function due to its common usage in periodic data. The sine of -0.8 radians is approximately -0.72.
Output: [-0.72]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
The data seems to be generated by a periodic function. Assuming it follows a common periodic pattern such as a sine or cosine wave, I’ll use the sine function for estimation. First, I normalize the input to align with the expected period of the sine function. Given the input [-2.4], I’ll compute the sine value and then scale it appropriately.
Step-by-step:
1. Assume a typical sine function with amplitude and frequency adjustments: ‘y = A * sin(B * x + C) + D‘.
2. Without knowing exact values of A, B, C, and D, assume A = 1, B = 1, C = 0, and D = 0 for simplicity: ‘y = sin(x)‘.
3. Compute sine for normalized input: ‘sin(-2.4)‘. Using calculator:
sin(-2.4) ≈ -0.67546
Result: Rounding to two decimal places: -0.68
Output: [-0.68]
————— **Overall Loss** —————
3.961
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-1.8] [2.4] [-2.4] [1.9] [-1.5] [-0.7] [-0.4] [1.8] [-1.3] [-0.8]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function.
**** The model outputs: ****
[[-1.] [0.7] [-0.7] [1.] [-1.] [-0.6] [-0.4] [1.] [-1.] [-0.7]]
**** The target outputs: ****[[1.] [2.7] [1.3] [2.9] [1.] [1.4] [1.6] [3.] [1.] [1.3]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the ’New Pattern Descriptions’. The model uses the ’New Pattern Descriptions’ should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don’t have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Optimization Output** —————
Reasoning:
The current model appears to be using a periodic function based on the pattern in the outputs. However, the outputs are not matching the target values, which suggests that the periodic function is an incorrect assumption or is not properly tuned for this data. The target outputs are all positive and range between 1 and 3, which suggests that the function generating these outputs may have a different form or additional components compared to what the current model is using. Given that the outputs should ideally be between 1 and 3, a possible optimization could be that the model should take into account a bias and scaling factor. One way to achieve this could be through a combination of a sinusoidal function and a linear transformation to ensure values fall within the target range.
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.5 * sin(0.8 * input) + 2.
This should ensure that the output values range from 1 to 3.
————— **Updated Theta** —————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.5 * sin(0.8 * input) + 2.
This should ensure that the output values range from 1 to 3.

Training History: Sinusoidal Regression – Step 2

Epoch 0 - Step 2
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.5 * sin(0.8 * input) + 2.
This should ensure that the output values range from 1 to 3.
** Input: **
[-2.8]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Forward Pass Output** —————
Explanations:
1. The given input is -1.4.
2. Compute the inner value of the sine function: 0.8 * -1.4 = -1.12.
3. Compute the sine of -1.12: sin(-1.12) ≈ -0.901.
4. Apply the transformation: 1.5 * -0.901 + 2 ≈ -1.352 + 2 = 0.65.
5. The output is rounded to two decimal places.
Output: [0.65]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
Step 1: Identify the input value which is 1.8.
Step 2: Use the transformation equation Output = 1.5 * sin(0.8 * input) + 2.
Step 3: Calculate the value inside the sine function: 0.8 * 1.8 = 1.44.
Step 4: Calculate the sine of this value: sin(1.44).
Step 5: Multiply the sine result by 1.5.
Step 6: Add 2 to the result from step 5.
Step 7: Round the final result to two decimal points. Calculation:
0.8 * 1.8 = 1.44
sin(1.44) ≈ 0.99
1.5 * 0.99 = 1.485
1.485 + 2 = 3.485
Rounded value = 3.49
Output: [3.49]
————— **Overall Loss** —————
0.30000000000000004
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-2.8] [-0.3] [1.8] [0.5] [2.2] [2.5] [3.] [-0.6] [1.2] [-1.4]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.5 * sin(0.8 * input) + 2.
This should ensure that the output values range from 1 to 3.
**** The model outputs: ****
[[0.8] [1.6] [3.5] [2.6] [3.5] [3.4] [3.] [1.3] [3.2] [0.6]]
**** The target outputs: ****
[[1.6] [1.7] [3.] [2.4] [2.8] [2.6] [2.2] [1.5] [2.9] [1.]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Optimization Output** —————
Reasoning:
Upon examining the current model outputs and their corresponding target outputs, it appears the current pattern description involving a combination of a sinusoidal function and a linear transformation is not yielding satisfactory results. The model outputs are often either too high or too low compared to the target values, indicating the need for more fine-tuning of our model. The new pattern descriptions should consider adjusting the amplitude, frequency, and bias term to better align the outputs with the targets. Observing the target outputs suggests a bias term close to 1.5 with a slightly dampened amplitude might be needed. This adjustment could get the model outputs closer to the desired range and alignment. Let's revise the current periodic function approach by adjusting the amplitude parameter from 1.5 to 1.2 and keeping the bias close to 1.5. We will also tune the frequency to 0.9 for better alignment with the target values:
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.9 * input) + 1.5.
This adjustment aims to improve the alignment of output values with the target range (1 to 3).
————— **Updated Theta** —————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.9 * input) + 1.5.
This adjustment aims to improve the alignment of output values with the target range (1 to 3).

Training History: Sinusoidal Regression – Step3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.9 * input) + 1.5.
This adjustment aims to improve the alignment of output values with the target range (1 to 3).
** Input: **
[-1.9]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
1. Identify the transformation function: Output = 1.2 * sin(0.9 * input) + 1.5.
2. Input value: 1.9.
3. Calculate 0.9 * 1.9 = 1.71.
4. Calculate sin(1.71) ≈ 0.99.
5. Calculate 1.2 * 0.99 ≈ 1.19.
6. Final Output: 1.19 + 1.5 = 2.69.
Output: [2.69]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. The given input is 1.3.
2. According to the transformation function: Output = 1.2 * sin(0.9 * input) + 1.5.
3. Calculate the intermediate value: 0.9 * 1.3 = 1.17.
4. Calculate the sine of this intermediate value: sin(1.17) ≈ 0.92 (using a calculator or sine table).
5. Multiply the sine value by 1.2: 1.2 * 0.92 = 1.104.
6. Add 1.5 to the result: 1.104 + 1.5 = 2.604.
7. Round the final output to two decimal places: 2.60.
Output: [2.60]
———— **Overall Loss** ————
0.602
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-1.9] [1.2] [1.9] [1.2] [-1.2] [3.] [-1.2] [1.3] [-1.7] [-2.5]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.9 * input) + 1.5.
This adjustment aims to improve the alignment of output values with the target range (1 to 3).
**** The model outputs: ****
[[0.3] [2.6] [1.2] [2.6] [0.4] [2.] [0.4] [2.6] [0.3] [0.6]]
**** The target outputs: ****
[[1.1] [2.9] [2.9] [2.9] [1.1] [2.2] [1.1] [3.] [1.] [1.4]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the ’New Pattern Descriptions’. The model uses the ’New Pattern Descriptions’ should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don’t have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model outputs indicate that the output values are still deviating significantly from the target values. This suggests that the previous adjustments for amplitude, frequency, and bias were not optimal enough. Specifically, the model outputs are predominantly lower than the desired target values, indicating that the output needs a higher bias term, potentially better alignment in the sinusoidal component’s amplitude, and possibly adapting the frequency further for an effective transformation.
Considering the previous adjustments and observing the new discrepancy, an increased bias term and a slight adjustment in the amplitude might offer closer alignment with the target distribution. Therefore, a more refined strategy would be to further increase the bias to 1.8, adjust the amplitude slightly to 1.3, and reevaluate the frequency to 1.0.
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.3 * sin(1.0 * input) + 1.8.
This further adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
———— **Updated Theta** ————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.3 * sin(1.0 * input) + 1.8.
This further adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.

Training History: Sinusoidal Regression – Step 4

Epoch 0 - Step 4
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.3 * sin(1.0 * input) + 1.8.
This further adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
** Input: **
[-2.2]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Forward Pass Output** —————
Explanations:
- The given input is 0.
- Using the transformation formula: Output = 1.3 * sin(1.0 * input) + 1.8.
- Substituting the input value: Output = 1.3 * sin(1.0 * 0) + 1.8.
- Calculating the sine value: sin(0) = 0.
- Applying the transformation: Output = 1.3 * 0 + 1.8 = 1.8.
Output: [1.80]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
1. Given the input of 1.8.
2. Calculate the sine of the input: sin(1.8).
3. Multiply the sine value by 1.3: 1.3 * sin(1.8).
4. Add 1.8 to the result from step 3.
Using a calculator:
sin(1.8) ≈ 0.97385
1.3 * 0.97385 ≈ 1.266005
1.266005 + 1.8 ≈ 3.066005
Output: [3.07]
————— **Overall Loss** —————
0.055000000000000003
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-2.2] [2.] [1.] [0.1] [1.8] [1.5] [1.8] [-1.1] [0.] [1.5]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.3 * sin(1.0 * input) + 1.8.
This further adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
**** The model outputs: ****
[[0.8] [3.] [2.9] [1.9] [3.1] [3.1] [3.1] [0.6] [1.8] [3.1]]
**** The target outputs: ****
[[1.2] [2.9] [2.8] [2.1] [3.] [3.] [3.] [1.1] [2.] [3.]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the 'New Pattern Descriptions'. The model uses the 'New Pattern Descriptions' should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
"' Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don't have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
————— **Optimization Output** —————
Reasoning:
The current model is still not performing optimally, as evidenced by discrepancies between the model outputs and the target outputs. Notably, some model outputs are higher than necessary, especially for higher input values. This indicates that while the increase in bias has helped, the amplitude or the frequency might need finer adjustments. Given that previous attempts have not aligned closely enough, we might need to reduce the amplitude slightly and change the frequency for a better fit. Previous frequency 1.0 might be slightly misaligned; hence we try reducing that to 0.8 and keeping a modest amplitude to ensure variability remains within check.
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.8 * input) + 1.8.
This adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
————— **Updated Theta** —————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.8 * input) + 1.8.
This adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.

Training History: Sinusoidal Regression – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.8 * input) + 1.8.
This adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
** Input: **
[0.3]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
1. Take the input value: 0.0.
2. Apply the transformation: Output = 1.2 * sin(0.8 * input) + 1.8.
3. Calculate the sine function: sin(0.8 * 0.0) = sin(0) = 0.
4. Calculate the product: 1.2 * 0 = 0.
5. Add the constant: 0 + 1.8 = 1.8.
Output: [1.80]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. Identify the input value: 1.7.
2. Use the given transformation formula to calculate the output:
Output = 1.2 * sin(0.8 * 1.7) + 1.8.
3. Calculate the value inside the sine function: 0.8 * 1.7 = 1.36.
4. Compute the sine of 1.36: sin(1.36) ≈ 0.9776.
5. Multiply the sine value by 1.2: 1.2 * 0.9776 ≈ 1.173.
6. Add 1.8 to the result: 1.173 + 1.8 ≈ 2.973.
7. Round the result to two decimal points: 2.97.
Output: [2.97]
———— **Overall Loss** ————
0.158
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[0.3] [-2.3] [0.7] [-2.1] [0.2] [0.] [-2.3] [1.7] [0.2] [-2.2]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.2 * sin(0.8 * input) + 1.8.
This adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
**** The model outputs: ****
[[2.1] [0.6] [2.4] [0.6] [2.] [1.8] [0.6] [3.] [2.] [0.6]]
**** The target outputs: ****
[[2.3] [1.2] [2.7] [1.1] [2.2] [2.] [1.2] [3.] [2.2] [1.2]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the ’New Pattern Descriptions’. The model uses the ’New Pattern Descriptions’ should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don’t have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is currently underperforming, particularly noticeable with the lower-than-expected outputs. The discrepancy points towards the need to adjust the amplitude and possibly the frequency further. Additionally, the model’s bias term seems to be correctly helping center the output but might still need a bit of fine-tuning to adjust for deviations. Hence, to better approximate the target outputs, we will slightly increase the frequency and adjust the amplitude. The previous frequency of 0.8 will be increased to 0.9 to see if it aligns the periodic component better. The amplitude will be increased slightly to 1.3, considering the higher-than-needed target values.
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.3 * sin(0.9 * input) + 1.8.
This adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.
———— **Updated Theta** ————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.3 * sin(0.9 * input) + 1.8.
This adjustment aims to better align the output values with the target range (1 to 3) and enhance prediction accuracy.

Training History: Sinusoidal Regression – Step 29

********** Epoch 2 - Step 9 **********
————— **Forward Pass Prompt** —————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.05 * sin(0.98 * input) + 2.0.
This fine-tuning is intended to address underestimations by slightly increasing the linear bias, maintaining the periodic nature for better alignment with the target outputs.
** Input: **
[2.6]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Forward Pass Output** —————
Explanations:
1. Calculate the inner function of the sine: 0.98 * 1.4 = 1.372.
2. Find the sine of the result: sin(1.372) ≈ 0.9803.
3. Multiply by the factor: 1.05 * 0.9803 ≈ 1.029.
4. Add the linear bias: 1.029 + 2.0 = 3.029.
Output: [3.03]

Many outputs for the rest data points ...

————— **Overall Loss** —————
0.003000000000000001
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[2.6] [2.9] [2.9] [0.1] [1.4] [-2.5] [2.1] [-2.4] [-0.8] [-1.6]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.05 * sin(0.98 * input) + 2.0.
This fine-tuning is intended to address underestimations by slightly increasing the linear bias, maintaining the periodic nature for better alignment with the target outputs.
**** The model outputs: ****
[[2.6] [2.3] [2.3] [2.1] [3.] [1.3] [2.9] [1.2] [1.3] [1.]]
**** The target outputs: ****
[[2.6] [2.3] [2.3] [2.1] [3.] [1.4] [2.8] [1.3] [1.3] [1.]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the ’New Pattern Descriptions’. The model uses the ’New Pattern Descriptions’ should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don’t have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Optimization Output** —————
Reasoning:
Analyzing the new set of inputs, model outputs, and target outputs:
1. For input [2.6], the model output is 2.6, matching the target output exactly.
2. For inputs [2.9] and [2.9], the model outputs are both 2.3, exactly matching the target outputs.
3. For input [0.1], the model output is 2.1, matching the target output exactly.
4. For input [1.4], the model output is 3.0, matching the target output exactly.
5. For input [-2.5], the model output is 1.3, while the target output is 1.4 – slight underestimation.
6. For input [2.1], the model output is 2.9, while the target output is 2.8 – slight overestimation.
7. For input [-2.4], the model output is 1.2, while the target output is 1.3 – slight underestimation.
8. For input [-0.8], the model output is 1.3, matching the target output exactly.
9. For input [-1.6], the model output is 1.0, matching the target output exactly.
From this analysis:
- Six out of ten inputs have model outputs matching the target outputs exactly.
- The model slightly underestimates results for inputs [-2.5] and [-2.4].
- The model slightly overestimates the result for input [2.1]. Reviewing the current pattern descriptions:
- The previous adjustment to the linear bias from 1.95 to 2.0 helped mitigate underestimations by lifting the overall predictions.
- However, slight underestimations still occur for negative inputs while an overestimation appears for a positive input. To balance these observations, I propose slightly adjusting the linear bias down a tiny step to correct the overestimations and adjusting the amplitude scaling factor to fine-tune the response to negative inputs:
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.06 * sin(0.98 * input) + 1.95.
This fine-tuning is intended to correct minor underestimations for negative inputs and slight overestimations for positive inputs, thereby maintaining overall alignment with the target outputs.
————— **Updated Theta** —————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.06 * sin(0.98 * input) + 1.95.
This fine-tuning is intended to correct minor underestimations for negative inputs and slight overestimations for positive inputs, thereby maintaining overall alignment with the target outputs.

Training History: Sinusoidal Regression – Step 30

Epoch 2 - Step 10
———— **Forward Pass Prompt** ————
You are the model. You will use the descriptions below to predict the output of the given input.
**** Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.06 * sin(0.98 * input) + 1.95.
This fine-tuning is intended to correct minor underestimations for negative inputs and slight overestimations for positive inputs, thereby maintaining overall alignment with the target outputs.
** Input: **
[-0.2]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[Your output MUST be in REAL NUMBER ROUNDED TO TWO DECIMAL POINTS; make necessary assumptions if needed; it MUST be in the same format as the Input]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
1. Calculate the sine of 0.98 times the input: sin(0.98 * 1) = sin(0.98) ≈ 0.830
2. Multiply the result by 1.06: 1.06 * 0.830 ≈ 0.880
3. Add 1.95 to the product: 0.880 + 1.95 ≈ 2.83
Output: [2.83]

Many outputs for the rest data points ...

———— **Overall Loss** ————
0.008000000000000004
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Pattern Descriptions below produced the outputs of the given inputs. You are given the target outputs, please optimize the Pattern Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data): ****
[[-0.2] [1.1] [1.9] [0.7] [-2.7] [-0.9] [1.7] [2.8] [1.] [-0.3]]
**** Current Pattern Descriptions: ****
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a periodic function. You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.06 * sin(0.98 * input) + 1.95.
This fine-tuning is intended to correct minor underestimations for negative inputs and slight overestimations for positive inputs, thereby maintaining overall alignment with the target outputs.
**** The model outputs: ****
[[1.7] [2.9] [3.] [2.6] [1.4] [1.1] [3.] [2.4] [2.8] [1.6]]
**** The target outputs: ****
[[1.8] [2.9] [3.] [2.7] [1.6] [1.2] [3.] [2.4] [2.8] [1.7]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please optimize the model by improving the ’New Pattern Descriptions’. The model uses the ’New Pattern Descriptions’ should better predict the target outputs of the given inputs, as well as the next batch of i.i.d. input data from the same distribution. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. NOTE: both the model and you can only operate on the numerical precision of one decimal points! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Pattern Descriptions by yourself; please show your work; note that you don’t have access to computer]
New Pattern Descriptions:
[put your new descriptions here; MUST be specific and concrete; ****MUST provide the exact value of the parameters if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the new set of inputs, model outputs, and target outputs:
1. For input [-0.2], the model output is 1.7, while the target output is 1.8 – slight underestimation.
2. For input [1.1], the model output is 2.9, matching the target output exactly.
3. For input [1.9], the model output is 3.0, matching the target output exactly.
4. For input [0.7], the model output is 2.6, while the target output is 2.7 – slight underestimation.
5. For input [-2.7], the model output is 1.4, while the target output is 1.6 – notable underestimation.
6. For input [-0.9], the model output is 1.1, while the target output is 1.2 – slight underestimation.
7. For input [1.7], the model output is 3.0, matching the target output exactly.
8. For input [2.8], the model output is 2.4, matching the target output exactly.
9. For input [1.0], the model output is 2.8, matching the target output exactly.
10. For input [-0.3], the model output is 1.6, while the target output is 1.7 – slight underestimation.
From this analysis:
- Five out of ten inputs have model outputs matching the target outputs exactly.
- The model slightly underestimates results for inputs [-0.2], [0.7], [-2.7], [-0.9], and [-0.3]. Reviewing the current pattern descriptions:
- The previous adjustment to the linear bias to 1.95 aimed to correct underestimations while also adjusting the amplitude scaling factor slightly improved estimations but slight underestimations still persist.
While adjusting the linear bias further down might correct some underestimations, the predominant deviation seems to involve negative inputs (less positive contributions). Also, ensuring periodic factor scaling stability:
New Pattern Descriptions:
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.07 * sin(0.98 * input) + 1.97.
This fine-tuning is intended to correct minor underestimations for negative inputs and ensuring overall alignment with the target outputs.
———— **Updated Theta** ————
You are designed to do regression, i.e., to predict the output of any given input. Both input and output are real numbers. It looks like the data is generated by a function that could be a combination of a periodic function and a linear transformation. Specifically, the model should use the transformation:
Output = 1.07 * sin(0.98 * input) + 1.97.
This fine-tuning is intended to correct minor underestimations for negative inputs and ensuring overall alignment with the target outputs.

M.4 Two Blobs Regression (LLama-3-70B without prior)

Training History: Two Blobs Classification – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0.
**** Input: ****
[-2.178 9.99]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: Since the model is not trained, I will make an assumption that the data is uniformly distributed across the 2-D plane. Based on this assumption, I will assign equal probabilities to both classes.
Output: [0.50 0.50]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: Since the model is a binary classifier, I'll make an assumption about the decision boundary. Let's assume the decision boundary is a linear line with a slope of -1 and passes through the origin. This means that points above the line belong to class 1 and points below the line belong to class 2. Given the input [-3.616, 7.818], we can calculate the dot product of the input with the normal vector of the decision boundary (which is [1, -1]). The dot product is -3.616 + 7.818 = 4.202, which is positive, so the point is above the decision boundary. Therefore, the point is more likely to belong to class 1.
Output: [0.70, 0.30]
———— **Training Accuracy** ————
0.5
———— **Overall Loss** ————
0.4302008151499229
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-2.178 9.99] [-3.111 10.867] [-4.234 8.452] [-3.616 7.818] [5.155 2.487] [-2.625 8.713] [3.75 1.157] [-2.523 7.957] [-3.348 8.705] [3.569 2.456]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0.
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[0.3 0.7] [0.99 0.01] [0.7 0.3] [0.7 0.3] [0.85 0.15] [0.5 0.5] [0.2 0.8] [0.2 0.8] [0.3 0.7] [0.65 0.35]]
**** The targets ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [1. 0.] [1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.]]
Please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The goal is to update the model descriptions to improve the classification accuracy. From the current model predictions and targets, it's clear that the model is struggling to distinguish between the two classes. Upon visual inspection of the input data, it appears that the data points with $x > 0$ tend to belong to class 2, while those with $x < 0$ tend to belong to class 1. Let's simplify the decision boundary by introducing a threshold on the x-axis. If $x > 0$, assign a high probability to class 2, otherwise assign a high probability to class 1.
New Model Descriptions:
If $x > 0$, output [0.0, 1.0]. If $x < 0$, output [1.0, 0.0].
———— **Updated Theta** ————
If $x > 0$, output [0.0, 1.0]. If $x < 0$, output [1.0, 0.0].

Training History: Two Blobs Classification – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, output [0.0, 1.0]. If x < 0, output [1.0, 0.0].
**** Input: ****
[-2.252 8.94]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations: Since x > 0, I will output [0.0, 1.0].
Output: [0.00, 1.00]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Given the input [6.784, 2.607], since x > 0, the output will be [0.0, 1.0].
Output: [0.00, 1.00]
———— **Training Accuracy** ————
1.0
———— **Overall Loss** ————
-4.9999999446126456e-09
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-2.252 8.94] [3.578 2.447] [-2.449 11.478] [-3.211 8.687] [6.784 2.607] [4.9 2.755] [4.653 3.427] [-1.594 9.343] [-2.702 9.316] [-1.044 8.789]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, output [0.0, 1.0]. If x < 0, output [1.0, 0.0].
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.] [0. 1.] [0. 1.] [1. 0.] [1. 0.] [1. 0.]]
**** The targets ([class 1 prob. class 2 prob.]): **** [[1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.] [0. 1.] [0. 1.] [1. 0.] [1. 0.] [1. 0.]]
Please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model descriptions are already quite simple and accurate, as they correctly classify all the input data points. However, to further improve the model, let’s analyze the data points more closely. We notice that the data points with y < 8.5 tend to belong to class 2, while those with y > 8.5 tend to belong to class 1. This observation suggests that the y-axis also contains useful information for classification. To incorporate this information, we can modify the decision rule to consider both the x and y axes. If x > 0, we still assign a high probability to class 2. If x < 0, we check the y value: if y < 8.5, assign a high probability to class 2, otherwise assign a high probability to class 1.
New Model Descriptions:
If x > 0, output [0.0, 1.0]. If x < 0, if y < 8.5, output [0.0, 1.0], otherwise output [1.0, 0.0].
———— **Updated Theta** ————
If x > 0, output [0.0, 1.0]. If x < 0, if y < 8.5, output [0.0, 1.0], otherwise output [1.0, 0.0].

Training History: Two Blobs Classification – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, output [0.0, 1.0]. If x < 0, if y < 8.5, output [0.0, 1.0], otherwise output [1.0, 0.0].
**** Input: ****
[-2.267 7.101]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Since x > 0, output [0.0, 1.0].
Output: [0.00, 1.00]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Since x < 0, we need to check the value of y. y is 7.551 which is less than 8.5. Therefore, the output should be [0.0, 1.0].
Output: [0.00, 1.00]
———— **Training Accuracy** ————
0.8
———— **Overall Loss** ————
1.8420680703952368
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-2.267 7.101] [3.96 2.205] [5.453 0.742] [-2.901 7.551] [-3.186 9.626] [-4.428 8.988] [5.723 3.027] [-3.054 9.125] [4.563 2.314] [-3.837 9.211]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, output [0.0, 1.0]. If x < 0, if y < 8.5, output [0.0, 1.0], otherwise output [1.0, 0.0].
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[0. 1.] [0. 1.] [0. 1.] [0. 1.] [1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.]]
**** The targets ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [0. 1.] [0. 1.] [1. 0.] [1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.]]
Please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model descriptions are still quite simple, but they make some mistakes in the current batch of data. Let’s analyze the data points more closely. We notice that the data points with x > 0 and y < 2.5 tend to belong to class 2, while those with x < 0 and y > 9 tend to belong to class 1. This observation suggests that the y-axis contains more useful information for classification, especially when x > 0. To incorporate this information, we can modify the decision rule to consider both the x and y axes more carefully. If x > 0, we check the y value: if y < 2.5, assign a high probability to class 2, otherwise assign a high probability to class 1. If x < 0, we also check the y value: if y > 9, assign a high probability to class 1, otherwise assign a high probability to class 2.
New Model Descriptions:
If x > 0, if y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 9.0, output [1.0, 0.0], otherwise output [0.0, 1.0].
———— **Updated Theta** ————
If x > 0, if y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 9.0, output [1.0, 0.0], otherwise output [0.0, 1.0].

Training History: Two Blobs Classification – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 9.0, output [1.0, 0.0], otherwise output [0.0, 1.0].
**** Input: ****
[-3.988 8.294]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Since x > 0 and y < 2.5, according to the rules, the output should be [0.0, 1.0].
Output: [0.00, 1.00]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Since x = 5.113 > 0, we check the value of y.
y = 1.9 < 2.5, so we output [0.0, 1.0].
Output: [0.00, 1.00]
———— **Training Accuracy** ————
0.7
———— **Overall Loss** ————
2.763102108092855
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-3.988 8.294] [-5.129 9.836] [3.403 0.653] [-3.522 9.329] [3.665 2.76] [-2.417 7.027] [5.113 1.9] [-3.318 8.513] [-3.925 8.594] [-3.417 7.602]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 9.0, output [1.0, 0.0], otherwise output [0.0, 1.0].
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [0. 1.] [0. 1.] [0. 1.]]
**** The targets ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [1. 0.] [1. 0.]]
Please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current batch of data, we notice that the current model descriptions are mostly correct, but there are some mistakes. For example, the point [-3.318 8.513] is misclassified, and it should belong to class 1. This suggests that the threshold for y-axis in the x < 0 region should be lowered. Additionally, we observe that the points with x > 0 and y > 2.5 tend to belong to class 1, and the points with x > 0 and y < 2.5 tend to belong to class 2. This suggests that the threshold for y-axis in the x > 0 region should be raised.
New Model Descriptions:
If x > 0, if y < 3.0, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 8.0, output [1.0, 0.0], otherwise output [0.0, 1.0].
———— **Updated Theta** ————
If x > 0, if y < 3.0, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 8.0, output [1.0, 0.0], otherwise output [0.0, 1.0].

Training History: Two Blobs Classification – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 3.0, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 8.0, output [1.0, 0.0], otherwise output [0.0, 1.0].
**** Input: ****
[4.698 0.83]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
x < 0, y > 8.0, so output [1.0, 0.0].
Output: [1.00, 0.00]

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
Since x > 0 (4.698 > 0), we check if y < 3.0 (0.83 < 3.0). This is true, so the output is [0.0, 1.0].
Output: [0.00, 1.00]
———— **Training Accuracy** ————
0.8
———— **Overall Loss** ————
1.8420680703952368
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[4.698 0.83] [-2.3 7.055] [5.462 3.87] [5.594 2.625] [5.162 2.27] [-1.697 10.371] [4.193 2.83] [-3.11 8.723] [3.262 1.035] [-2.412 9.983]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 3.0, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 8.0, output [1.0, 0.0], otherwise output [0.0, 1.0].
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[0. 1.] [0. 1.] [1. 0.] [0. 1.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.]]
**** The targets ([class 1 prob. class 2 prob.]): ****
[[0. 1.] [1. 0.] [0. 1.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.]]
Please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current batch of data, we notice that the current model descriptions are mostly correct, but there are some mistakes. For example, the point [-2.3 7.055] is misclassified, and it should belong to class 1. This suggests that the threshold for y-axis in the x < 0 region should be lowered. Additionally, we observe that the points with x > 0 and y > 2.5 tend to belong to class 1, and the points with x > 0 and y < 2.5 tend to belong to class 2. This suggests that the threshold for y-axis in the x > 0 region is correct.
New Model Descriptions:
If x > 0, if y < 3.0, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 7.5, output [1.0, 0.0], otherwise output [0.0, 1.0].
———— **Updated Theta** ————
If x > 0, if y < 3.0, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if y > 7.5, output [1.0, 0.0], otherwise output [0.0, 1.0].

Training History: Two Blobs Classification – Step 81

Epoch 8 - Step 1
————— **Forward Pass Prompt** —————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise output [1.0, 0.0].
**** Input: ****
[-2.178 9.99]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Forward Pass Output** —————
Explanations:
Since x > 0 and y < 1.7, the output is [0.0, 1.0].
Output: [0.00, 1.00]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
Since x > 0, we check the conditions for x > 0.
x = 3.569 > 3.0 and y = 2.456 < 2.2, so the output is [0.0, 1.0].
Output: [0.00, 1.00]
————— **Training Accuracy** —————
1.0
————— **Overall Loss** —————
-4.999999944612645 6e-09
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-2.178 9.99] [-3.111 10.867] [-4.234 8.452] [-3.616 7.818] [5.155 2.487] [-2.625 8.713] [3.75 1.157] [-2.523 7.957] [-3.348 8.705] [3.569 2.456]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise output [1.0, 0.0].
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [1. 0.] [1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.]]
**** The targets ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [1. 0.] [1. 0.] [1. 0.] [0. 1.] [1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.]]
Please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.

Training History: Two Blobs Classification – Step 81 (continued)

Optimization Output

Reasoning:
The model is performing well, but I noticed that the points with x < -3.0 and y > 9.5 are more likely to belong to class 1. By analyzing the data, I see that the points with x < -3.0 and y > 9.5 are consistently classified as class 1. So, I'll add a new rule to output [1.0, 0.0] in this case. Additionally, I observed that the points with x > 4.0 and y < 2.5 are more likely to belong to class 2. I'll add a new rule to output [0.0, 1.0] for these points.
New Model Descriptions:
If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise output [1.0, 0.0].

Updated Theta

If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise output [1.0, 0.0].

Training History: Two Blobs Classification – Step 82

Epoch 8 - Step 2
————— **Forward Pass Prompt** —————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise output [1.0, 0.0].
**** Input: ****
[-2.252 8.94]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY A PURE probability vector, where each value is between 0.0 and 1.0 WITH TWO DECIMAL POINTS; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Forward Pass Output** —————
Explanations:
Since x < 0, we check the conditions for x < 0.
x < -3.5 and y > 8.5, which is true.
Output: [1.00, 0.00]

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
Since x < 0, we check the conditions for x < 0.
x = -2.449, y = 11.478.
The condition x < -2.0 and y > 10.0 is satisfied.
So, the output is [1.0, 0.0].
Output: [1.00, 0.00]
————— **Training Accuracy** —————
1.0
————— **Overall Loss** —————
-4.9999999446126456e-09
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted how likely the given inputs belong to a class. You are given the target values, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-2.252 8.94] [3.578 2.447] [-2.449 11.478] [-3.211 8.687] [6.784 2.607] [4.9 2.755] [4.653 3.427] [-1.594 9.343] [-2.702 9.316] [-1.044 8.789]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output a vector containing two probabilities such that each corresponds to how likely the data belongs to each class, i.e., [class 1 prob. class 2 prob.]. The sum of the vector MUST be 1.0. If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise output [1.0, 0.0].
**** The model predictions ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.] [0. 1.] [0. 1.] [1. 0.] [1. 0.] [1. 0.]]
**** The targets ([class 1 prob. class 2 prob.]): ****
[[1. 0.] [0. 1.] [1. 0.] [1. 0.] [0. 1.] [0. 1.] [0. 1.] [1. 0.] [1. 0.] [1. 0.]]
Please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Both the model and you MUST ONLY operate on the numerical precision of THREE decimal points. You are bad with numerical calculations, so be extra careful! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.

Training History: Two Blobs Classification – Step 82 (continued)

Optimization Output

Reasoning:
The model is performing well, but I noticed that the points with x > 4.5 and y < 2.8 are more likely to belong to class 2. By analyzing the data, I see that the points with x > 4.5 and y < 2.8 are consistently classified as class 2. So, I'll add a new rule to output [0.0, 1.0] in this case. Additionally, I observed that the points with x < -2.5 and y > 9.2 are more likely to belong to class 1. I'll add a new rule to output [1.0, 0.0] for these points.
New Model Descriptions:
If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 9.2, output [1.0, 0.0], otherwise output [1.0, 0.0].

Updated Theta

If x > 0, if y < 1.7, output [0.0, 1.0], otherwise if y < 2.0 and x > 3.0, output [0.0, 1.0], otherwise if y < 2.8 and x > 4.0, output [0.0, 1.0], otherwise if y < 2.5 and x > 3.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 4.5, output [0.0, 1.0], otherwise if y < 3.5 and x > 5.0, output [0.0, 1.0], otherwise if y < 3.0 and x > 3.5, output [0.0, 1.0], otherwise if y < 4.3 and x <= 4.0, output [1.0, 0.0], otherwise if y < 4.3 and x > 4.0, output [0.0, 1.0], otherwise if y > 9.5, output [1.0, 0.0], otherwise if x > 4.5 and y < 3.0, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 3.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.5 and y < 2.8, output [0.0, 1.0], otherwise if x > 5.0 and y < 3.5, output [0.0, 1.0], otherwise if x > 3.5 and y < 2.2, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise if x > 4.0 and y < 1.2, output [0.0, 1.0], otherwise if x > 5.0 and y < 2.3, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.0, output [0.0, 1.0], otherwise if x > 4.0 and y < 2.5, output [0.0, 1.0], otherwise output [1.0, 0.0]. If x < 0, if x < -3.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 10.0, output [1.0, 0.0], otherwise if x < -2.5 and y > 10.0, output [1.0, 0.0], otherwise if x < -1.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 7.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.5 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 8.0, output [1.0, 0.0], otherwise if x < -4.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -4.5 and y > 8.5, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.8, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.4, output [1.0, 0.0], otherwise if x < -1.5 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.0, output [1.0, 0.0], otherwise if x < -2.0 and y > 9.2, output [1.0, 0.0], otherwise if x < -3.0 and y > 9.5, output [1.0, 0.0], otherwise if x < -2.5 and y > 9.2, output [1.0, 0.0], otherwise output [1.0, 0.0].

M.5 Two Circles Regression (LLama-3-70B without prior)

Training History: Two Circles Classification without Prior – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1.
**** Input: ****
[-0.146 -1.01]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Since I don't have any information about the decision boundary, I will make an assumption that the decision boundary is the x=y line.
The point (0.89, 0.397) is below the x=y line, so it belongs to class 0.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: Since I don't have any prior knowledge about the classification model, I'll assume a simple linear decision boundary. Let's assume the decision boundary is x + y = 0. Then, I'll plug in the input values to determine the class label.
Plugging in x = -0.146 and y = -1.01, we get -0.146 + (-1.01) = -1.156 < 0. Since the result is negative, the point lies below the decision boundary.
Output: 0
———— **Training Accuracy** ————
0.4
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.146 -1.01] [0.224 -0.929] [0.89 0.397] [0.811 -0.56] [-0.873 0.475] [0.27 0.172] [0.08 0.261] [0.518 0.943] [-0.502 -0.788] [-0.069 -0.252]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1.
**** The model predictions: ****
[[0] [1] [0] [1] [1] [1] [0] [1] [0] [0]]
**** The targets: ****
[[0] [0] [0] [0] [0] [1] [1] [0] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current model predictions and targets, I notice that the model is not performing well, especially for points with positive y-values. Upon visual inspection of the input data, I observe that points with y-values greater than 0 tend to belong to class 1, while points with y-values less than 0 tend to belong to class 0. This suggests that the y-coordinate is a strong feature for classification. Therefore, I will update the model descriptions to incorporate this insight.
New Model Descriptions:
If y > 0, then class label = 1. Otherwise, class label = 0.
———— **Updated Theta** ————
If y > 0, then class label = 1. Otherwise, class label = 0.

Training History: Two Circles Classification without Prior – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If y > 0, then class label = 1. Otherwise, class label = 0.
**** Input: ****
[-0.305 0.936]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: Since y > 0, the class label is 1.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: Given the input point [-0.735 -0.614], we need to determine the class label based on the condition provided. The condition states that if y > 0, then the class label is 1, otherwise it is 0. In this case, y = -0.614, which is less than 0.
Output: 0
———— **Training Accuracy** ————
0.3
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.305 0.936] [0.298 -0.037] [0.396 0.931] [-0.735 -0.614] [-0.339 0.006] [-0.253 -0.015] [0.096 -0.248] [1.005 0.048] [-0.329 0.046] [-0.051 -0.372]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If y > 0, then class label = 1. Otherwise, class label = 0.
**** The model predictions: ****
[[1] [0] [1] [0] [1] [0] [0] [1] [1] [0]]
**** The targets: ****
[[0] [1] [0] [0] [1] [1] [1] [0] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning: Analyzing the current model predictions and targets, I notice that the model is not performing well, especially for points with x-values greater than 0. Upon visual inspection of the input data, I observe that points with x-values greater than 0.3 tend to belong to class 1, while points with x-values less than 0.3 tend to belong to class 0. This suggests that the x-coordinate is also a strong feature for classification. Therefore, I will update the model descriptions to incorporate this insight.
New Model Descriptions:
If x > 0.3, then class label = 1. If x <= 0.3 and y > 0, then class label = 1. Otherwise, class label = 0.
———— **Updated Theta** ————
If x > 0.3, then class label = 1. If x <= 0.3 and y > 0, then class label = 1. Otherwise, class label = 0.

Training History: Two Circles Classification without Prior – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x > 0.3, then class label = 1. If x <= 0.3 and y > 0, then class label = 1. Otherwise, class label = 0.
**** Input: ****
[0.275 0.163]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations: Since x = 0.403 > 0.3, we can directly determine the class label.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. The input is [0, -1.012].
2. Since x = 0 is not greater than 0.3, move to the next condition.
3. Since x <= 0.3 and y = -1.012 is not greater than 0, the class label is 0.
Output: 0
———— **Training Accuracy** ————
0.6
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[0.275 0.163] [0. -1.012] [-1.065 -0.126] [0.133 0.22] [-0.317 -0.931] [0.403 0.96] [0.013 0.996] [0.147 -0.178] [-0.028 0.422] [-0.233 -0.186]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x > 0.3, then class label = 1. If x <= 0.3 and y > 0, then class label = 1. Otherwise, class label = 0.
**** The model predictions: ****
[[1] [0] [0] [1] [0] [1] [1] [0] [1] [0]]
**** The targets: ****
[[1] [0] [0] [1] [0] [0] [0] [1] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current model predictions and targets, I notice that the model is not performing well, especially for points with x-values around 0. Upon visual inspection of the input data, I observe that points with y-values greater than 0.2 tend to belong to class 1, while points with y-values less than 0.2 tend to belong to class 0. This suggests that the y-coordinate is also a strong feature for classification. Therefore, I will update the model descriptions to incorporate this insight.
New Model Descriptions:
If x > 0.3, then class label = 1. If x <= 0.3 and y > 0.2, then class label = 1. Otherwise, class label = 0.
———— **Updated Theta** ————
If x > 0.3, then class label = 1. If x <= 0.3 and y > 0.2, then class label = 1. Otherwise, class label = 0.

Training History: Two Circles Classification without Prior – Step 4

Epoch 0 - Step 4
————— **Forward Pass Prompt** —————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x > 0.3, then class label = 1. If x <= 0.3 and y > 0.2, then class label = 1. Otherwise, class label = 0.
**** Input: ****
[-0.811 0.529]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Forward Pass Output** —————
Explanations: Since x = 0.943 > 0.3, the class label is 1.
Output: 1

Many outputs for the rest data points ...

————— **Forward Pass Output** —————
Explanations:
Since x = -0.995, which is less than or equal to 0.3, we need to check the value of y.
Since y = -0.025, which is less than or equal to 0.2, we conclude that the class label is 0.
Output: 0
————— **Training Accuracy** —————
0.2
————— **Optimization Prompt** —————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.811 0.529] [-0.036 0.323] [-0.63 0.698] [-0.173 -0.246] [0.943 -0.297] [0.1 -0.288] [-0.995 -0.025] [0.666 -0.666] [0.24 -0.039] [0.542 0.887]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x > 0.3, then class label = 1. If x <= 0.3 and y > 0.2, then class label = 1. Otherwise, class label = 0.
**** The model predictions: ****
[[1] [1] [1] [0] [1] [0] [0] [1] [0] [1]]
**** The targets: ****
[[0] [1] [0] [1] [0] [1] [0] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
————— **Optimization Output** —————
Reasoning:
Analyzing the current model predictions and targets, I notice that the model is not performing well, especially for points with x-values around 0. Upon visual inspection of the input data, I observe that points with x-values less than -0.5 tend to belong to class 0, while points with x-values greater than 0.5 tend to belong to class 0 as well. This suggests that the x-coordinate has a non-linear relationship with the class label. I also notice that points with y-values greater than 0.5 tend to belong to class 0, while points with y-values less than -0.2 tend to belong to class 1. This suggests that the y-coordinate is also important for classification. Therefore, I will update the model descriptions to incorporate these insights.
New Model Descriptions:
If x < -0.5 or x > 0.5, then class label = 0. If y > 0.5, then class label = 0. If y < -0.2, then class label = 1. Otherwise, class label = 1.
————— **Updated Theta** —————
If x < -0.5 or x > 0.5, then class label = 0. If y > 0.5, then class label = 0. If y < -0.2, then class label = 1. Otherwise, class label = 1.

Training History: Two Circles Classification without Prior – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x < -0.5 or x > 0.5, then class label = 0. If y > 0.5, then class label = 0. If y < -0.2, then class label = 1. Otherwise, class label = 1.
**** Input: ****
[-0.99 -0.325]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Since x = -1.059 < -0.5, class label = 0.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. Check if x < -0.5 or x > 0.5. Since 0.167 is between -0.5 and 0.5, this condition is not met.
2. Check if y > 0.5. Since 0.235 is less than 0.5, this condition is not met.
3. Check if y < -0.2. Since 0.235 is greater than -0.2, this condition is not met.
4. Since none of the above conditions are met, the class label is 1.
Output: 1
———— **Training Accuracy** ————
0.8
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.99 -0.325] [-0.825 0.438] [0.188 0.308] [-1.059 0.217] [-0.188 -0.184] [-0.322 0.084] [0.892 0.515] [-0.287 0.017] [0.167 0.235] [-0.573 -0.845]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x < -0.5 or x > 0.5, then class label = 0. If y > 0.5, then class label = 0. If y < -0.2, then class label = 1. Otherwise, class label = 1.
**** The model predictions: ****
[[0] [0] [0] [0] [1] [1] [0] [1] [1] [1]]
**** The targets: ****
[[0] [0] [1] [0] [1] [1] [0] [1] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current model predictions and targets, I notice that the model is still not performing well. Upon visual inspection of the input data, I observe that points with y-values around 0 tend to be misclassified. This suggests that the y-coordinate threshold of 0.5 is too high. I also notice that points with x-values around -0.2 tend to belong to class 1, while points with x-values around 0.2 tend to belong to class 0. This suggests that the x-coordinate threshold of 0.5 is too high as well. Therefore, I will update the model descriptions to incorporate these insights.
New Model Descriptions:
If x < -0.4 or x > 0.4, then class label = 0. If y > 0.3, then class label = 0. If y < -0.2, then class label = 1. Otherwise, class label = 1.
———— **Updated Theta** ————
If x < -0.4 or x > 0.4, then class label = 0. If y > 0.3, then class label = 0. If y < -0.2, then class label = 1. Otherwise, class label = 1.

Training History: Two Circles Classification without Prior – Step 40

Epoch 3 - Step 10
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x < -0.75 or x > 0.95, then class label = 0. If y > 0.78, then class label = 0. If y < -0.42, then class label = 0. Otherwise, if x < 0.44, then class label = 1, else class label = 0.
**** Input: ****
[-0.101 0.309]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Since 0.919 > 0.95, class label = 0.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. Check if x < -0.75 or x > 0.95. Since -0.655 is not in this range, move to the next step.
2. Check if y > 0.78. Since -0.649 is not in this range, move to the next step.
3. Check if y < -0.42. Since -0.649 is in this range, class label = 0.
Output: 0
———— **Training Accuracy** ————
1.0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.101 0.309] [-1.011 -0.165] [0.161 -0.229] [1.02 -0.061] [0.225 0.076] [-0.024 -0.333] [-0.655 -0.649] [-1.058 0.15] [0.366 -0.098] [0.919 0.08]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x < -0.75 or x > 0.95, then class label = 0. If y > 0.78, then class label = 0. If y < -0.42, then class label = 0. Otherwise, if x < 0.44, then class label = 1, else class label = 0.
**** The model predictions: ****
[[1] [0] [1] [0] [1] [1] [0] [0] [1] [0]]
**** The targets: ****
[[1] [0] [1] [0] [1] [1] [0] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current model predictions and targets, I observe that the model is performing well, but there is still room for improvement. Upon visual inspection of the input data, I notice that the current x-coordinate threshold of 0.44 seems to be effective, and the y-coordinate thresholds of 0.78 and -0.42 are reasonable. However, I observe that some points with x-values around -0.6 and y-values around -0.65 are misclassified, which suggests that the model is not capturing the class boundary accurately in this region. I also notice that points with x-values around 0.2 and y-values around 0.1 are correctly classified, which suggests that the model is performing well in this region. Additionally, I observe that the current x-coordinate threshold of 0.95 seems to be too aggressive, as it misclassifies some points with x-values around 0.9.
New Model Descriptions:
If x < -0.75 or x > 0.92, then class label = 0. If y > 0.76, then class label = 0. If y < -0.41, then class label = 0. Otherwise, if x < 0.43, then class label = 1, else class label = 0.
———— **Updated Theta** ————
If x < -0.75 or x > 0.92, then class label = 0. If y > 0.76, then class label = 0. If y < -0.41, then class label = 0. Otherwise, if x < 0.43, then class label = 1, else class label = 0.

Training History: Two Circles Classification without Prior – Step 41

Epoch 4 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x < -0.75 or x > 0.92, then class label = 0. If y > 0.76, then class label = 0. If y < -0.41, then class label = 0. Otherwise, if x < 0.43, then class label = 1, else class label = 0.
**** Input: ****
[-0.146 -1.01]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given input [-0.873, 0.475].
Since x < -0.75, class label = 0.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. Check if x < -0.75 or x > 0.92. Since 0.08 is not in this range, move to the next step.
2. Check if y > 0.76. Since 0.261 is not in this range, move to the next step.
3. Check if y < -0.41. Since 0.261 is not in this range, move to the next step.
4. Since x < 0.43, class label = 1.
Output: 1
———— **Training Accuracy** ————
1.0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.146 -1.01] [0.224 -0.929] [0.89 0.397] [0.811 -0.56] [-0.873 0.475] [0.27 0.172] [0.08 0.261] [0.518 0.943] [-0.502 -0.788] [-0.069 -0.252]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. If x < -0.75 or x > 0.92, then class label = 0. If y > 0.76, then class label = 0. If y < -0.41, then class label = 0. Otherwise, if x < 0.43, then class label = 1, else class label = 0.
**** The model predictions: ****
[[0] [0] [0] [0] [0] [1] [1] [0] [0] [1]]
**** The targets: ****
[[0] [0] [0] [0] [0] [1] [1] [0] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Analyzing the current model predictions and targets, I observe that the model is performing well, but there is still room for improvement. Upon visual inspection of the input data, I notice that the current x-coordinate threshold of 0.43 seems to be effective, and the y-coordinate thresholds of 0.76 and -0.41 are reasonable. However, I observe that some points with x-values around -0.5 and y-values around -0.7 are misclassified, which suggests that the model is not capturing the class boundary accurately in this region. I also notice that points with x-values around 0.2 and y-values around 0.2 are correctly classified, which suggests that the model is performing well in this region. Additionally, I observe that the current x-coordinate threshold of 0.92 seems to be too aggressive, as it misclassifies some points with x-values around 0.8. New Model Descriptions: If x < -0.75 or x > 0.88, then class label = 0. If y > 0.74, then class label = 0. If y < -0.39, then class label = 0. Otherwise, if x < 0.42, then class label = 1, else class label = 0.
———— **Updated Theta** ————
If x < -0.75 or x > 0.88, then class label = 0. If y > 0.74, then class label = 0. If y < -0.39, then class label = 0. Otherwise, if x < 0.42, then class label = 1, else class label = 0.

M.6 Two Circles Regression (LLama-3-70B with prior)

Training History: Two Circles Classification with Prior – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle.
**** Input: ****
[-0.146 -1.01]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
The decision boundary is a circle, so we need to find the distance of the input point [-0.069, -0.252] from the origin (0,0). The distance is sqrt((-0.069)^2 + (-0.252)^2) = 0.264. Since the distance is less than 1, the point is inside the circle.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
The decision boundary is a circle. A point [x y] is classified as 0 if it is inside the circle, and 1 if it is outside the circle. The center of the circle is assumed to be at [0 0]. The radius of the circle is assumed to be 1 (a unit circle). The distance of the point [0.89 0.397] from the center of the circle is sqrt(0.89^2 + 0.397^2) = 1. Since the distance is equal to the radius, the point is on the circle. In this case, it is assumed that the point is classified as 0.
Output: 0
———— **Training Accuracy** ————
0.5
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.146 -1.01] [0.224 -0.929] [0.89 0.397] [0.811 -0.56] [-0.873 0.475] [0.27 0.172] [0.08 0.261] [0.518 0.943] [-0.502 -0.788] [-0.069 -0.252]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle.
**** The model predictions: ****
[[0] [0] [0] [1] [0] [0] [0] [1] [0] [0]]
**** The targets: ****
[[0] [0] [0] [0] [0] [1] [1] [0] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is not performing well, as it misclassifies 4 out of 10 samples. The decision boundary is a circle, but the current model predictions do not seem to follow a circular pattern. To improve the model, I will try to find a better circle that separates the classes. I will plot the data points on the 2-D plane and try to find a circle that separates the classes. After visualizing the data, I notice that the points (0.27, 0.172) and (0.08, 0.261) are misclassified, and they are close to each other. I will try to find a circle that passes through the midpoint of these two points and is centered at a point that is closer to the origin. Let the center of the circle be (a, b) and the radius be r. The midpoint of the two points is (0.175, 0.2165). I will set the center of the circle to be (0.175, 0.2165). The radius can be found by calculating the distance from the center to one of the points, say (0.27, 0.172). The radius is approximately 0.1.
New Model Descriptions:
If (x - 0.175)^2 + (y - 0.2165)^2 <= 0.01, then output 1, else output 0.
———— **Updated Theta** ————
If (x - 0.175)^2 + (y - 0.2165)^2 <= 0.01, then output 1, else output 0.

Training History: Two Circles Classification with Prior – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x - 0.175)^2 + (y - 0.2165)^2 <= 0.01, then output 1, else output 0.
**** Input: ****
[-0.305 0.936]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output: [ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Calculate the value of (x - 0.175)^2 + (y - 0.2165)^2 = (-0.051 - 0.175)^2 + (-0.372 - 0.2165)^2 = 0.1061. Since 0.1061 > 0.01, the point is outside the circle.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
The given input point is [-0.329, 0.046]. To determine the class label, we need to check if this point lies within the circle with the decision boundary. The circle's center is at (0.175, 0.2165) and the radius is sqrt(0.01) = 0.1. Let's calculate the distance of the point from the center:
Distance = sqrt((x - 0.175)^2 + (y - 0.2165)^2)
= sqrt((-0.329 - 0.175)^2 + (0.046 - 0.2165)^2)
= sqrt((-0.504)^2 + (-0.1705)^2)
= sqrt(0.253696 + 0.02914025)
= sqrt(0.28283625)
= 0.532
Since the distance (0.532) is greater than the radius (0.1), the point lies outside the circle.
Output: 0
———— **Training Accuracy** ————
0.4
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.305 0.936] [0.298 -0.037] [0.396 0.931] [-0.735 -0.614] [-0.339 0.006] [-0.253 -0.015] [0.096 -0.248] [1.005 0.048] [-0.329 0.046] [-0.051 -0.372]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x - 0.175)^2 + (y - 0.2165)^2 <= 0.01, then output 1, else output 0.
**** The model predictions: ****
[[0] [0] [0] [0] [0] [0] [0] [0] [0] [0]]
**** The targets: ****
[[0] [1] [0] [0] [1] [1] [1] [0] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is not performing well, as it misclassifies 7 out of 10 samples. The current decision boundary is a circle centered at (0.175, 0.2165) with a radius of 0.1, but it does not seem to separate the classes well. I will re-examine the data and try to find a better circle that separates the classes. I notice that the points (0.298, -0.037) and (-0.339, 0.006) are misclassified, and they are close to each other. I will try to find a circle that passes through the midpoint of these two points and is centered at a point that is closer to the origin. Let the center of the circle be (a, b) and the radius be r. The midpoint of the two points is (-0.0205, -0.0155). I will set the center of the circle to be (-0.0205, -0.0155). The radius can be found by calculating the distance from the center to one of the points, say (0.298, -0.037). The radius is approximately 0.35.
New Model Descriptions:
If (x + 0.0205)^2 + (y + 0.0155)^2 <= 0.1225, then output 1, else output 0.
———— **Updated Theta** ————
If (x + 0.0205)^2 + (y + 0.0155)^2 <= 0.1225, then output 1, else output 0.

Training History: Two Circles Classification with Prior – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x + 0.0205)^2 + (y + 0.0155)^2 <= 0.1225, then output 1, else output 0.
**** Input: ****
[0.275 0.163]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Calculate the value of the decision boundary equation: (0.403 + 0.0205)^2 + (0.96 + 0.0155)^2 = 0.4235^2 + 0.9755^2 = 0.179 + 0.952 = 1.131. Since 1.131 > 0.1225, the point is outside the circle.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
We are given a point (0,-1.012) on the 2-D plane. We need to check if the point lies inside the circle defined by the decision boundary. Substituting the values in the equation: (0 + 0.0205)^2 + (-1.012 + 0.0155)^2 <= 0.1225.
Calculating the values: (0.0205)^2 + (-0.9965)^2 <= 0.1225.
Calculating the values: 0.00042025 + 0.99242025 <= 0.1225.
Calculating the values: 0.9928405 <= 0.1225.
The condition is not satisfied.
Output: 0
———— **Training Accuracy** ————
0.7
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[0.275 0.163] [0. -1.012] [-1.065 -0.126] [0.133 0.22] [-0.317 -0.931] [0.403 0.96] [0.013 0.996] [0.147 -0.178] [-0.028 0.422] [-0.233 -0.186]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x + 0.0205)^2 + (y + 0.0155)^2 <= 0.1225, then output 1, else output 0.
**** The model predictions: ****
[[1] [0] [0] [0] [0] [0] [0] [0] [0] [1]]
**** The targets: ****
[[1] [0] [0] [1] [0] [0] [0] [1] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is not performing well, as it misclassifies 4 out of 10 samples. The current decision boundary is a circle centered at (-0.0205, -0.0155) with a radius of approximately 0.35, but it does not seem to separate the classes well. I will re-examine the data and try to find a better circle that separates the classes. I notice that the points (0.133, 0.22) and (0.147, -0.178) are misclassified, and they are close to each other. I will try to find a circle that passes through the midpoint of these two points and is centered at a point that is closer to the origin. Let the center of the circle be (a, b) and the radius be r. The midpoint of the two points is (0.14, 0.021). I will set the center of the circle to be (0.14, 0.021). The radius can be found by calculating the distance from the center to one of the points, say (0.133, 0.22). The radius is approximately 0.235.
New Model Descriptions:
If (x - 0.14)^2 + (y - 0.021)^2 <= 0.055225, then output 1, else output 0.
———— **Updated Theta** ————
If (x - 0.14)^2 + (y - 0.021)^2 <= 0.055225, then output 1, else output 0.

Training History: Two Circles Classification with Prior – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x - 0.14)^2 + (y - 0.021)^2 <= 0.055225, then output 1, else output 0.
**** Input: ****
[-0.811 0.529]
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Calculate the distance from the point to the center of the circle: (0.24 - 0.14)^2 + (-0.039 - 0.021)^2 = 0.0064 + 0.0036 = 0.0100
Since 0.0100 <= 0.055225, the point is inside the circle.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
We need to check if the point (-0.811, 0.529) lies inside the circle (x - 0.14)^2 + (y - 0.021)^2 <= 0.055225.
First, calculate the left side of the inequality: (-0.811 - 0.14)^2 + (0.529 - 0.021)^2 = 0.671^2 + 0.508^2 = 0.449441 + 0.258064 = 0.707505.
Since 0.707505 > 0.055225, the point (-0.811, 0.529) lies outside the circle.
Output: 0
———— **Training Accuracy** ————
0.7
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.811 0.529] [-0.036 0.323] [-0.63 0.698] [-0.173 -0.246] [0.943 -0.297] [0.1 -0.288] [-0.995 -0.025] [0.666 -0.666] [0.24 -0.039] [0.542 0.887]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x - 0.14)^2 + (y - 0.021)^2 <= 0.055225, then output 1, else output 0.
**** The model predictions: ****
[[0] [0] [0] [0] [0] [0] [0] [0] [1] [0]]
**** The targets: ****
[[0] [1] [0] [1] [0] [1] [0] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is not performing well, as it misclassifies 6 out of 10 samples. The current decision boundary is a circle centered at (0.14, 0.021) with a radius of approximately 0.235, but it does not seem to separate the classes well. I will re-examine the data and try to find a better circle that separates the classes. I notice that the points (-0.036, 0.323), (-0.173, -0.246), and (0.1, -0.288) are misclassified, and they are all close to the origin. I will try to find a circle that passes through the origin and is centered at a point that is closer to the origin. Let the center of the circle be (a, b) and the radius be r. I will set the center of the circle to be (0, 0). The radius can be found by calculating the distance from the center to one of the points, say (-0.173, -0.246). The radius is approximately 0.292.
New Model Descriptions:
If (x)^2 + (y)^2 <= 0.085328, then output 1, else output 0.
———— **Updated Theta** ————
If (x)^2 + (y)^2 <= 0.085328, then output 1, else output 0.

Training History: Two Circles Classification with Prior – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x)^2 + (y)^2 <= 0.085328, then output 1, else output 0.
**** Input: ****
[-0.99 -0.325]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given input [-1.059, 0.217], calculate (x)^2 + (y)^2 = (-1.059)^2 + (0.217)^2 = 1.123.
Since 1.123 > 0.085328, it does not satisfy the condition.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
We are given a point [-0.825, 0.438] on a 2-D plane.
We need to check if the point lies inside the circle or not.
The equation of the circle is (x)^2 + (y)^2 <= 0.085328.
Let's calculate the value of (x)^2 + (y)^2 for the given point. (-0.825)^2 + (0.438)^2 = 0.680625 + 0.191844 = 0.872469.
Since 0.872469 > 0.085328, the point lies outside the circle.
Output: 0
———— **Training Accuracy** ————
0.7
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.99 -0.325] [-0.825 0.438] [0.188 0.308] [-1.059 0.217] [-0.188 -0.184] [-0.322 0.084] [0.892 0.515] [-0.287 0.017] [0.167 0.235] [-0.573 -0.845]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x)^2 + (y)^2 <= 0.085328, then output 1, else output 0.
**** The model predictions: ****
[[0] [0] [0] [0] [1] [0] [0] [1] [0] [0]]
**** The targets: ****
[[0] [0] [1] [0] [1] [1] [0] [1] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is not performing well, as it misclassifies 5 out of 10 samples. The current decision boundary is a circle centered at the origin with a radius of approximately 0.292, but it does not seem to separate the classes well. I will re-examine the data and try to find a better circle that separates the classes. I notice that the points (0.188, 0.308), (-0.188, -0.184), and (0.167, 0.235) are misclassified, and they are all close to the origin. I will try to find a circle that passes through the origin and is centered at a point that is closer to the origin. Let the center of the circle be (a, b) and the radius be r. I will set the center of the circle to be (0, 0). The radius can be found by calculating the distance from the center to one of the points, say (0.167, 0.235). The radius is approximately 0.283. However, this radius is not sufficient to separate the classes well. I will try to find a better radius by examining the other misclassified points. The point (-0.322, 0.084) is also misclassified, and it is closer to the origin than the previous point. The distance from the center to this point is approximately 0.334. I will use this distance as the radius.
New Model Descriptions:
If (x)^2 + (y)^2 <= 0.111556, then output 1, else output 0.
———— **Updated Theta** ————
If (x)^2 + (y)^2 <= 0.111556, then output 1, else output 0.

Training History: Two Circles Classification with Prior – Step 40

Epoch 3 - Step 10
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x)^2 + (y)^2 <= 0.45, then output 1, else output 0.
**** Input: ****
[-0.101 0.309]
Please give your output strictly in the following format:
“‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
“‘
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations:
Given point [-1.011 -0.165], calculate the sum of squares: (-1.011)^2 + (-0.165)^2 = 1.044721 > 0.45.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
The given input point is [-0.024, -0.333].
To determine the class label, we need to check if the point is inside the circle or not.
The equation of the circle is (x)^2 + (y)^2 <= 0.45.
Let’s calculate the value of this equation for the given point: (-0.024)^2 + (-0.333)^2 = 0.011056 + 0.110889 = 0.121945 <= 0.45.
Since the result is true, the point lies inside the circle.
Output: 1
———— **Training Accuracy** ————
0.9
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.101 0.309] [-1.011 -0.165] [0.161 -0.229] [1.02 -0.061] [0.225 0.076] [-0.024 -0.333] [-0.655 -0.649] [-1.058 0.15] [0.366 -0.098] [0.919 0.08]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x)^2 + (y)^2 <= 0.45, then output 1, else output 0.
**** The model predictions: ****
[[1] [0] [1] [0] [1] [1] [1] [0] [1] [0]]
**** The targets: ****
[[1] [0] [1] [0] [1] [1] [0] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. DON’T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
“‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
”’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is not performing well, as it misclassifies the 7th sample. The current decision boundary is a circle centered at the origin with a radius of 0.45. The 7th sample is far from the origin, but the model still classifies it as 1. This means the radius of the circle is too small. I will increase the radius to reduce the misclassification error.
New Model Descriptions:
If (x)^2 + (y)^2 <= 0.65, then output 1, else output 0.
———— **Updated Theta** ————
If (x)^2 + (y)^2 <= 0.65, then output 1, else output 0.

Training History: Two Circles Classification with Prior – Step 41

Epoch 4 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x)^2 + (y)^2 <= 0.65, then output 1, else output 0.
**** Input: ****
[-0.146 -1.01]
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
First, calculate the value of (x)^2 + (y)^2: (-0.146)^2 + (-1.01)^2 = 0.021236 + 1.0201 = 1.041336.
Since 1.041336 > 0.65, the point is outside the circle.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
The input point is [-0.502 -0.788].
To determine the class label, we need to check if the point satisfies the condition (x)^2 + (y)^2 <= 0.65.
Here, x = -0.502 and y = -0.788. (x)^2 = (-0.502)^2 = 0.252004, and (y)^2 = (-0.788)^2 = 0.621984.
Now, (x)^2 + (y)^2 = 0.252004 + 0.621984 = 0.874988.
Since 0.874988 <= 0.65 is False, the point lies outside the circle.
Output: 0
———— **Training Accuracy** ————
1.0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. data on 2-D plane: [x y]): ****
[[-0.146 -1.01] [0.224 -0.929] [0.89 0.397] [0.811 -0.56] [-0.873 0.475] [0.27 0.172] [0.08 0.261] [0.518 0.943] [-0.502 -0.788] [-0.069 -0.252]]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a point on a 2-D plane [x y]; you need to output the class label, i.e., an integer in the set 0, 1. The decision boundary is a circle. If (x)^2 + (y)^2 <= 0.65, then output 1, else output 0.
**** The model predictions: ****
[[0] [0] [0] [0] [0] [1] [1] [0] [0] [1]]
**** The targets: ****
[[0] [0] [0] [0] [0] [1] [1] [0] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. DON'T use symbolic representation for the model! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here; MUST be concise and concrete; ****MUST PROVIDE THE EXACT VALUE OF THE PARAMETERS if the descriptions potentially involve unknown or learnable parameters!!!****]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is performing well, as it correctly classifies all samples. The decision boundary is a circle centered at the origin with a radius of 0.65, which seems to be a good fit for the data. No changes are needed.
New Model Descriptions:
If (x)^2 + (y)^2 <= 0.65, then output 1, else output 0.
———— **Updated Theta** ————
If (x)^2 + (y)^2 <= 0.65, then output 1, else output 0.

M.7 Text Classification (LLama-3-70B without prior)

Training History: Text Classification – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1.
**** Input: ****
fatherhood
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: Since the input is a name, "Margaret", I will assume it's a female name. Therefore, I will classify it as a female name.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The term "tomboy" is a noun that refers to a girl who exhibits characteristics or behaviors considered typical of a boy. It does not have any strong connotations or associations that would clearly link it to a specific class label. However, based on the context and common usage of the term, I will make an assumption that the class label 1 represents a more masculine or boy-like characteristic, and 0 represents a more feminine or girl-like characteristic. Given this assumption, I would classify "tomboy" as 1.
Output: 1
———— **Training Accuracy** ————
0.5
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. text): ****
['fatherhood', 'grandfather', 'tomboy', 'Shawn', 'Erma', 'Ada', 'Margaret', 'Caleb', 'Earl', 'Faye']
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1.
**** The model predictions: ****
[[1] [1] [1] [1] [1] [1] [1] [0] [1] [0]]
**** The targets: ****
[[0] [0] [1] [0] [1] [1] [1] [0] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
Upon examining the current model predictions and target labels, I notice that the model is not performing well, with 6 out of 10 predictions incorrect. The model seems to be biased towards predicting class label 1. I observe that the input terms can be categorized into two groups: male and female names, and non-name terms related to family relationships. The target labels suggest that the class label 0 corresponds to male names and non-name terms, while class label 1 corresponds to female names. To improve the model, I will create new decision rules that take into account these categories. I will update the descriptions to include more specific patterns and rules to differentiate between male and female names, as well as non-name terms.
New Model Descriptions:
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term is a non-name term related to family relationships (e.g., fatherhood, grandfather), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye), predict class label 1. Otherwise, predict class label 0.
———— **Updated Theta** ————
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term is a non-name term related to family relationships (e.g., fatherhood, grandfather), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye), predict class label 1. Otherwise, predict class label 0.

Training History: Text Classification – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term is a non-name term related to family relationships (e.g., fatherhood, grandfather), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye), predict class label 1. Otherwise, predict class label 0.
**** Input: ****
Lydia
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The input term "Marlene" is a female name.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The input term is "paternal grandma". The term "grandma" is a non-name term related to family relationships, and "paternal" is an adjective that means related to father. Therefore, according to the rules, the class label is 0.
Output: 0
———— **Training Accuracy** ————
0.9
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. text): ****
['Lydia', 'Steven', 'Ida', 'Lisa', 'Allan', 'Jane', 'paternal grandma', 'paternal grandad', 'Marlene', 'Darrell']
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term is a non-name term related to family relationships (e.g., fatherhood, grandfather), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye), predict class label 1. Otherwise, predict class label 0.
**** The model predictions: ****
[[1] [0] [1] [1] [0] [1] [0] [0] [1] [0]]
**** The targets: ****
[[1] [0] [1] [1] [0] [1] [1] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
Upon examining the current model predictions and target labels, I notice that the model is performing reasonably well, but there are still some errors. The model correctly predicts 8 out of 10 labels, with errors occurring for the terms "paternal grandma" and "paternal grandad". I realize that the current descriptions do not account for non-name terms that contain keywords related to family relationships, but are not exact matches (e.g., "paternal grandma" vs. "grandma"). To improve the model, I will update the descriptions to include more specific rules for non-name terms that contain keywords related to family relationships. I will also refine the rules for female names to reduce the likelihood of misclassification.
New Model Descriptions:
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. Otherwise, predict class label 0.
———— **Updated Theta** ————
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. Otherwise, predict class label 0.

Training History: Text Classification – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. Otherwise, predict class label 0.
**** Input: ****
Marissa
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations: The input term is "Maria", which is a female name.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
1. Check if the term ends with a suffix commonly associated with male names: Julian does not end with a suffix like -son or -er.
2. Check if the term contains keywords related to family relationships: Julian does not contain keywords like "father", "mother", "grand", "paternal", or "maternal".
3. Check if the term is a female name: Julian is not a female name.
4. Check if the term is a name that does not clearly indicate gender: Julian is a male name.
Output: 0
———— **Training Accuracy** ————
0.9
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. text): ****
[’Marissa’, ’Wallace’, ’Julian’, ’Adeline’, ’Jacob’, ’Dylan’, ’William’, ’Anne’, ’Maria’, ’sorority’]
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. Otherwise, predict class label 0.
**** The model predictions: ****
[[1] [0] [0] [1] [0] [0] [0] [1] [1] [0]]
**** The targets: ****
[[1] [0] [0] [1] [0] [0] [0] [1] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the ’New Model Descriptions’, which should have lower classification error both on the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
Upon examining the current model predictions and target labels, I notice that the model is performing well, but there is still one error occurring for the term "sorority". I realize that the current descriptions do not account for non-name terms that are clearly related to females or males, but are not names. To improve the model, I will update the descriptions to include more specific rules for non-name terms that are clearly related to females or males. I will also refine the rules for female names to reduce the likelihood of misclassification.
New Model Descriptions:
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl"), predict class label 1. Otherwise, predict class label 0.
———— **Updated Theta** ————
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl"), predict class label 1. Otherwise, predict class label 0.

Training History: Text Classification – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl"), predict class label 1. Otherwise, predict class label 0.
**** Input: ****
Natasha
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: Veronica is a female name, so predict class label 1.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The input term "Antonio" is a male name, and it does not contain any keywords related to family relationships. It does not end with a suffix commonly associated with male names, but it is clearly a male name.
Output: 0
———— **Training Accuracy** ————
1.0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. text): ****
['Natasha', 'Veronica', 'Dawn', 'Mamie', 'Darren', 'Monique', 'gynecological', 'Antonio', 'Eula', 'Henry']
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl"), predict class label 1. Otherwise, predict class label 0.
**** The model predictions: ****
[[1] [1] [1] [1] [0] [1] [1] [0] [1] [0]]
**** The targets: ****
[[1] [1] [1] [1] [0] [1] [1] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
Upon examining the current model predictions and target labels, I notice that the model is performing well, but there is no need for further improvement since all predictions match the target labels. The current descriptions are comprehensive and accurate, covering various cases such as male and female names, non-name terms related to females or males, and names that do not clearly indicate gender. Since the model is already performing perfectly, I will not make any changes to the descriptions.
New Model Descriptions:
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria, Natasha, Veronica, Dawn, Mamie, Monique, Eula), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl", "gynecological"), predict class label 1. Otherwise, predict class label 0.
———— **Updated Theta** ————
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria, Natasha, Veronica, Dawn, Mamie, Monique, Eula), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl", "gynecological"), predict class label 1. Otherwise, predict class label 0.

Training History: Text Classification – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria, Natasha, Veronica, Dawn, Mamie, Monique, Eula), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl", "gynecological"), predict class label 1. Otherwise, predict class label 0.
**** Input: ****
Danny
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The input term "Betty" is a female name.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The input term is "brethren". It does not end with a suffix commonly associated with male names, nor does it contain keywords related to family relationships. It is not a female name, and it does not clearly indicate gender. However, it is a term related to a group of males (brothers), so it is more associated with males than females.
Output: 0
———— **Training Accuracy** ————
1.0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given inputs. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. text): ****
['Danny', 'Betty', 'Calvin', 'Janet', 'Allison', 'missis', 'Brandon', 'Carl', 'Ben', 'brethren']
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is a term; you need to output the class label, i.e., an integer in the set 0, 1. If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria, Natasha, Veronica, Dawn, Mamie, Monique, Eula), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl", "gynecological"), predict class label 1. Otherwise, predict class label 0.
**** The model predictions: ****
[[0] [1] [0] [1] [1] [1] [0] [0] [0] [0]]
**** The targets: ****
[[0] [1] [0] [1] [1] [1] [0] [0] [0] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving the 'New Model Descriptions', which should have lower classification error both on the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, improve the Current Model Descriptions by yourself; please show your work; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
Upon examining the current model predictions and target labels, I notice that the model is performing well, with all predictions matching the target labels. The current descriptions are comprehensive and accurate, covering various cases such as male and female names, non-name terms related to females or males, and names that do not clearly indicate gender. However, I would like to update the female names list to include the new female names encountered in this batch, which are 'Betty', 'Janet', and 'Allison'. Additionally, I notice that the term 'missis' is a non-name term related to females, so I will add it to the list of non-name terms. Since the term 'brethren' is a non-name term related to males, I will add it to the list of non-name terms as well.
New Model Descriptions:
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria, Natasha, Veronica, Dawn, Mamie, Monique, Eula, Betty, Janet, Allison), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl", "gynecological", "missis"), predict class label 1. If the input term is a non-name term clearly related to males (e.g., "brethren"), predict class label 0. Otherwise, predict class label 0.
———— **Updated Theta** ————
If the input term ends with a suffix commonly associated with male names (e.g., -son, -er), predict class label 0. If the input term contains keywords related to family relationships (e.g., "father", "mother", "grand", "paternal", "maternal"), predict class label 0. If the input term is a female name (e.g., Erma, Ada, Faye, Lydia, Ida, Jane, Marlene, Marissa, Adeline, Anne, Maria, Natasha, Veronica, Dawn, Mamie, Monique, Eula, Betty, Janet, Allison), predict class label 1. If the input term is a name that does not clearly indicate gender (e.g., Leslie, Jordan), predict class label 0. If the input term is a non-name term clearly related to females (e.g., "sorority", "girl", "gynecological", "missis"), predict class label 1. If the input term is a non-name term clearly related to males (e.g., "brethren"), predict class label 0. Otherwise, predict class label 0.

M.8 Medical Image Classification (GPT-4o with prior)

Training History: Medical Image Classification (with Prior) – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is an image; you need to output the class label, i.e., an integer in the set 0, 1. The input is X-ray image for identifying pneumonia.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
The X-ray image shows signs of lung opacity and consolidation, which are indicative of pneumonia. The presence of these features suggests an abnormality consistent with pneumonia.
Output: 1
———— **Forward Pass Output** ————
Explanations:
The X-ray image shows signs of lung opacity and consolidation, which are indicative of pneumonia. The presence of these abnormalities suggests an infection in the lungs.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
The X-ray image shows the chest area. I analyzed the lung fields for any signs of opacity, consolidation, or other abnormalities that are indicative of pneumonia. The image appears to have clear lung fields without significant opacities or consolidations.
Output: 0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is an image; you need to output the class label, i.e., an integer in the set 0, 1. The input is X-ray image for identifying pneumonia.
**** The model predictions: ****
[[1] [1] [0] [1] [1] [0] [1] [1] [1] [1]]
**** The targets: ****
[[0] [0] [0] [1] [0] [0] [1] [0] [0] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model descriptions fail to adequately differentiate between pneumonia and non-pneumonia cases leading to incorrect predictions. Upon analyzing the images, I observed key indicators such as lung opacity, consolidations, and the presence of fluid. Enhanced model descriptions should include these visual features for improved accuracy.
New Model Descriptions:
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity or consolidations are present, or if there is fluid in the lungs.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity or fluid presence.
———— **Updated Theta** ————
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity or consolidations are present, or if there is fluid in the lungs.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity or fluid presence.

Training History: Medical Image Classification (with Prior) – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity or consolidations are present, or if there is fluid in the lungs.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity or fluid presence.

**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations:
The X-ray image shows clear lung fields without significant opacity or fluid presence. There are no visible consolidations or signs of lung opacity.
Output: 0
———— **Forward Pass Output** ————
Explanations:
The X-ray image shows clear lung fields without significant opacity or fluid presence. There are no visible consolidations or signs of lung opacity.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations:
The X-ray image shows significant lung opacity and consolidations, indicating the presence of fluid in the lungs.
Output: 1
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity or consolidations are present, or if there is fluid in the lungs.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity or fluid presence.

**** The model predictions: ****
[[1] [1] [0] [0] [1] [0] [1] [1] [1] [1]]
**** The targets: ****
[[1] [1] [0] [0] [0] [0] [1] [1] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model descriptions are still leading to some misclassifications, particularly it underestimates cases that show very mild or early signs of pneumonia. Addition of more granulated signs of pneumonia such as patchy infiltrates, mild interstitial markings or signs of hyperinflation can improve the performance. Also noting bilateral versus unilateral presentations can be a useful signal.
New Model Descriptions:
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, bilateral infiltrates, or mild interstitial markings are present.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, or other mentioned indicators of pneumonia.

———— **Updated Theta** ————
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, bilateral infiltrates, or mild interstitial markings are present.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, or other mentioned indicators of pneumonia.

Training History: Medical Image Classification (with Prior) – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, bilateral infiltrates, or mild interstitial markings are present.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, or other mentioned indicators of pneumonia.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The X-ray image shows clear lung fields without significant opacity, consolidations, fluid, bilateral infiltrates, or mild interstitial markings.
Output: 0
———— **Forward Pass Output** ————
Explanations: The X-ray image shows significant lung opacity and consolidations in both lungs. There are also signs of fluid presence and bilateral infiltrates.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The X-ray image shows significant lung opacity and consolidations. There are also indications of fluid presence and bilateral infiltrates.
Output: 1
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, bilateral infiltrates, or mild interstitial markings are present.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, or other mentioned indicators of pneumonia.
**** The model predictions: ****
[[1] [1] [1] [0] [1] [1] [1] [1] [0] [1]]
**** The targets: ****
[[1] [1] [0] [0] [0] [0] [0] [1] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels] "'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current descriptions are leading to over-identification of pneumonia (more false positives). Incorporating the presence of specific signs and ruling out other conditions can be helpful. Adding features like the presence of pleural effusion, absence of heart enlargement, and noting whether the opacity is localized (can be other pathologies) may help. Precision in observing these smaller details will aid the classification accuracy.
New Model Descriptions:
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, pleural effusion, bilateral infiltrates, or mild interstitial markings are present and not associated with heart enlargement.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, pleural effusion, or other mentioned indicators of pneumonia, or if opacities are localized and not typical for pneumonia.
———— **Updated Theta** ————
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, pleural effusion, bilateral infiltrates, or mild interstitial markings are present and not associated with heart enlargement.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, pleural effusion, or other mentioned indicators of pneumonia, or if opacities are localized and not typical for pneumonia.

Training History: Medical Image Classification (with Prior) – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, pleural effusion, bilateral infiltrates, or mild interstitial markings are present and not associated with heart enlargement.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, pleural effusion, or other mentioned indicators of pneumonia, or if opacities are localized and not typical for pneumonia.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The X-ray image shows significant lung opacity and consolidations in both lungs. There is no clear indication of heart enlargement. The presence of these features is consistent with pneumonia.
Output: 1
———— **Forward Pass Output** ————
Explanations: The X-ray image shows significant lung opacity and consolidations. There are also signs of fluid presence and bilateral infiltrates. These findings are consistent with pneumonia and are not associated with heart enlargement.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The X-ray image shows clear lungs without significant opacity, fluid presence, pleural effusion, or other indicators of pneumonia. There are no visible lung opacities, consolidations, or bilateral infiltrates.
Output: 0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if lung opacity, consolidations, fluid, pleural effusion, bilateral infiltrates, or mild interstitial markings are present and not associated with heart enlargement.
- Output 0 (No Pneumonia) if lungs appear clear without significant opacity, fluid presence, pleural effusion, or other mentioned indicators of pneumonia, or if opacities are localized and not typical for pneumonia.
**** The model predictions: ****
[[0] [0] [1] [1] [1] [1] [1] [1] [1] [0]]
**** The targets: ****
[[0] [0] [0] [0] [0] [1] [1] [1] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model still produces false positives. To refine the criteria, focus on more specific patterns indicative of pneumonia like lobar consolidation or "ground-glass" opacities. Exclude cases where opacities are likely due to external factors such as overlying soft tissue shadows or slight variations that are non-diagnostic.
New Model Descriptions:
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if clear indications of lobar consolidation, ground-glass opacities, fluid presence (excluding pleural effusion), bilateral infiltrates, or moderate interstitial markings are present and not related to heart enlargement.
- Output 0 (No Pneumonia) if lung fields appear clear without significant opacity, fluid, bilateral infiltrates, or other pneumonia indicators, or if features suggest alternative explanations (e.g., overlying soft tissue shadows).
———— **Updated Theta** ————
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if clear indications of lobar consolidation, ground-glass opacities, fluid presence (excluding pleural effusion), bilateral infiltrates, or moderate interstitial markings are present and not related to heart enlargement.
- Output 0 (No Pneumonia) if lung fields appear clear without significant opacity, fluid, bilateral infiltrates, or other pneumonia indicators, or if features suggest alternative explanations (e.g., overlying soft tissue shadows).

Training History: Medical Image Classification (with Prior) – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if clear indications of lobar consolidation, ground-glass opacities, fluid presence (excluding pleural effusion), bilateral infiltrates, or moderate interstitial markings are present and not related to heart enlargement.
- Output 0 (No Pneumonia) if lung fields appear clear without significant opacity, fluid, bilateral infiltrates, or other pneumonia indicators, or if features suggest alternative explanations (e.g., overlying soft tissue shadows).
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The X-ray image shows clear indications of bilateral infiltrates and moderate interstitial markings. There are no signs of heart enlargement that could explain these features.
Output: 1
———— **Forward Pass Output** ————
Explanations: The X-ray image shows clear lung fields without significant opacity, fluid presence, bilateral infiltrates, or other indicators of pneumonia. There are no clear signs of lobar consolidation, ground-glass opacities, or moderate interstitial markings.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The X-ray image shows clear indications of lobar consolidation and ground-glass opacities in the lung fields. These findings are consistent with pneumonia.
Output: 1
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if clear indications of lobar consolidation, ground-glass opacities, fluid presence (excluding pleural effusion), bilateral infiltrates, or moderate interstitial markings are present and not related to heart enlargement.
- Output 0 (No Pneumonia) if lung fields appear clear without significant opacity, fluid, bilateral infiltrates, or other pneumonia indicators, or if features suggest alternative explanations (e.g., overlying soft tissue shadows).
**** The model predictions: ****
[[0] [0] [1] [0] [1] [0] [0] [0] [1] [1]]
**** The targets: ****
[[1] [0] [1] [0] [1] [0] [0] [0] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning: The current model descriptions are missing some complexities encountered in the images. We observe some misclassifications indicating the need to differentiate between the types of opacities more clearly. Including signs such as air bronchograms and the pattern of unilateral versus bilateral presentation can aid in improving the description.
New Model Descriptions:
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if there are signs of lobar consolidation, ground-glass opacities, air bronchograms, fluid presence (excluding pleural effusion), bilateral infiltrates, or moderate interstitial markings.
- Output 0 (No Pneumonia) if lung fields appear clear without significant opacity or fluid, if opacities are limited to a single lung and are not of the typical pneumonia pattern, or if these indicators are due to overlying soft tissue shadows.
———— **Updated Theta** ————
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if there are signs of lobar consolidation, ground-glass opacities, air bronchograms, fluid presence (excluding pleural effusion), bilateral infiltrates, or moderate interstitial markings.
- Output 0 (No Pneumonia) if lung fields appear clear without significant opacity or fluid, if opacities are limited to a single lung and are not of the typical pneumonia pattern, or if these indicators are due to overlying soft tissue shadows.

Training History: Medical Image Classification (with Prior) – Step 50

Epoch 4 - Step 10
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if there are lobar consolidations, asymmetric or bilateral ground-glass opacities, air bronchograms, moderate to significant alveolar fluid, asymmetric infiltrates, patchy inflammations, moderate to severe pleural effusion, central or peripheral acute opacities, air-fluid levels, increasing opacities, pneumatocele formations, rapidly changing patterns in chronic lesions indicative of acute infection, or persistent cough-based findings.
- Output 0 (No Pneumonia) if lung fields are clear, minimal localized fluid, no distinct acute opacities, stable fibrotic scarring, solitary calcifications, old healed lesions, stable localized non-infectious anomalies, chronic interstitial fibrosis, chronic reticulonodular patterns without recent changes, stable chronic lesions without acute signs, or purely non-persistent anomalies.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The X-ray image shows lobar consolidations and asymmetric ground-glass opacities. There are also visible air bronchograms and moderate alveolar fluid. These findings are indicative of pneumonia.
Output: 1
———— **Forward Pass Output** ————
Explanations: The X-ray image shows clear lung fields without any significant opacities, consolidations, or fluid accumulations. There are no signs of air bronchograms, pleural effusion, or other acute changes indicative of pneumonia.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The X-ray image shows lobar consolidations and asymmetric ground-glass opacities. There are also visible air bronchograms and moderate alveolar fluid. These findings are indicative of pneumonia.
Output: 1
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if there are lobar consolidations, asymmetric or bilateral ground-glass opacities, air bronchograms, moderate to significant alveolar fluid, asymmetric infiltrates, patchy inflammations, moderate to severe pleural effusion, central or peripheral acute opacities, air-fluid levels, increasing opacities, pneumatocele formations, rapidly changing patterns in chronic lesions indicative of acute infection, or persistent cough-based findings.
- Output 0 (No Pneumonia) if lung fields are clear, minimal localized fluid, no distinct acute opacities, stable fibrotic scarring, solitary calcifications, old healed lesions, stable localized non-infectious anomalies, chronic interstitial fibrosis, chronic reticulonodular patterns without recent changes, stable chronic lesions without acute signs, or purely non-persistent anomalies.
**** The model predictions: ****
[[1] [0] [0] [0] [1] [1] [1] [0] [0] [1]]
**** The targets: ****
[[1] [0] [0] [1] [0] [1] [0] [0] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.

Training History: Medical Image Classification (with Prior) – Step 50 (continued)

——————— **Optimization Output** ———————
Reasoning:
The model missed some true cases of pneumonia and incorrectly identified some non-pneumonia cases. To improve the accuracy, refine the criteria to better identify pneumonia even in less obvious cases and differentiate between chronic stable conditions and acute infections.
New Model Descriptions:
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if there are lobar consolidations, asymmetric or bilateral ground-glass opacities, air bronchograms, moderate to significant alveolar fluid, asymmetric infiltrates, patchy inflammations, moderate to severe pleural effusion, central or peripheral acute opacities, air-fluid levels, increasing opacities, pneumatocele formations, rapidly changing patterns in chronic lesions indicative of acute infection, persistent cough-related findings, or recent onset of ground-glass opacities.
- Output 0 (No Pneumonia) if lung fields are clear, minimal localized fluid, no distinct acute opacities, stable fibrotic scarring, solitary calcifications, old healed lesions, stable localized non-infectious anomalies, chronic interstitial fibrosis, chronic reticulonodular patterns without recent changes, stable chronic lesions without acute signs, or absence of any acute infection markers.
——————— **Updated Theta** ———————
The input is an X-ray image. To classify:
- Output 1 (Pneumonia) if there are lobar consolidations, asymmetric or bilateral ground-glass opacities, air bronchograms, moderate to significant alveolar fluid, asymmetric infiltrates, patchy inflammations, moderate to severe pleural effusion, central or peripheral acute opacities, air-fluid levels, increasing opacities, pneumatocele formations, rapidly changing patterns in chronic lesions indicative of acute infection, persistent cough-related findings, or recent onset of ground-glass opacities.
- Output 0 (No Pneumonia) if lung fields are clear, minimal localized fluid, no distinct acute opacities, stable fibrotic scarring, solitary calcifications, old healed lesions, stable localized non-infectious anomalies, chronic interstitial fibrosis, chronic reticulonodular patterns without recent changes, stable chronic lesions without acute signs, or absence of any acute infection markers.

M.9 Medical Image Classification (GPT-4o without prior)

Training History: Medical Image Classification (without Prior) – Step 1

Epoch 0 - Step 1
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
You are designed to do binary classification. The input is an image; you need to output the class label, i.e., an integer in the set 0, 1.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
‘‘‘
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Forward Pass Output** ————
Explanations: The image appears to be a chest X-ray. I will analyze the image for any abnormalities or signs of disease that could indicate the class label.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The image appears to be a medical X-ray of a chest. The analysis involves examining the features and patterns in the image to determine the class label. Based on the visual inspection, the image shows certain characteristics that align with class 1.
Output: 1
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
You are designed to do binary classification. The input is an image; you need to output the class label, i.e., an integer in the set 0, 1.
**** The model predictions: ****
[[0] [0] [1] [1] [1] [0] [0] [1] [1] [1]]
**** The targets: ****
[[0] [0] [0] [1] [0] [0] [1] [0] [0] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the ’Current Model Descriptions’, which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous ’Optimization Step’ are provided, you can use the information from your last optimization step if it’s helpful. Limit your ’New Model Descriptions’ to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
‘‘‘
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don’t have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
’’’
Please ONLY reply according to this format, don’t give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model’s performance indicates significant misclassification. To improve, I must explicitly identify and use key visual features in the images that are indicative of the target classes. For instance, if the classification relates to medical diagnostics (e.g., chest X-rays for pneumonia), specific features like the presence of fluid, opacity, or other anomalies must be emphasized. Similarly, non-medical classifications (typical objects versus atypical ones) could require identification of distinct shapes, colors, or textural features. Analyzing the given images visually will guide the addition of pertinent descriptive rules that improve classification accuracy.
New Model Descriptions:
1. If the image contains prominent, clearly defined geometric or bodily structures and no unusual opacity or anomalies, output the class label 0.
2. If the image shows abnormal features such as irregular shapes, unexpected opacities, or non-uniform patterns that suggest pathology or anomaly, output the class label 1.
3. When distinguishing medical images specifically, check for fluid accumulation, granular textures, or any signs of abnormality in the expected anatomy; if these are present, output the class label 1.
———— **Updated Theta** ————
1. If the image contains prominent, clearly defined geometric or bodily structures and no unusual opacity or anomalies, output the class label 0.
2. If the image shows abnormal features such as irregular shapes, unexpected opacities, or non-uniform patterns that suggest pathology or anomaly, output the class label 1.
3. When distinguishing medical images specifically, check for fluid accumulation, granular textures, or any signs of abnormality in the expected anatomy; if these are present, output the class label 1.

Training History: Medical Image Classification (without Prior) – Step 2

Epoch 0 - Step 2
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
1. If the image contains prominent, clearly defined geometric or bodily structures and no unusual opacity or anomalies, output the class label 0.
2. If the image shows abnormal features such as irregular shapes, unexpected opacities, or non-uniform patterns that suggest pathology or anomaly, output the class label 1.
3. When distinguishing medical images specifically, check for fluid accumulation, granular textures, or any signs of abnormality in the expected anatomy; if these are present, output the class label 1.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The image shows irregular shapes and unexpected opacities. There are non-uniform patterns that suggest an anomaly.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The image appears to be a medical X-ray. There are irregular shapes and unexpected opacities present in the lung area, which suggest pathology or anomaly.
Output: 1
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
1. If the image contains prominent, clearly defined geometric or bodily structures and no unusual opacity or anomalies, output the class label 0.
2. If the image shows abnormal features such as irregular shapes, unexpected opacities, or non-uniform patterns that suggest pathology or anomaly, output the class label 1.
3. When distinguishing medical images specifically, check for fluid accumulation, granular textures, or any signs of abnormality in the expected anatomy; if these are present, output the class label 1.
**** The model predictions: ****
[[1] [1] [1] [1] [1] [1] [1] [1] [1] [1]]
**** The targets: ****
[[1] [1] [0] [0] [0] [0] [1] [1] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The current model is too sensitive to features that are indicating anomalies, leading to many false positives. A refined approach would include more granular analysis to distinguish normal variations from actual abnormalities. Specifically, enhancing the detection criteria to differentiate between minor variations and significant pathological markers is necessary. Reviewing more images, I will refine the descriptions to include clearer definitions of significant anomalies versus natural structures.
New Model Descriptions:
1. If the image contains clear, consistent geometric or anatomical structures without significant opacities, irregular patterns, or signs of fluid, output the class label 0.
2. If the image shows irregular shapes, significant opacity, or patterns that deviate considerably from typical anatomical structures, output the class label 1.
3. Examine for fluid accumulation, granular textures, and other significant deviations in expected anatomy. Minor irregularities that do not suggest severe pathology should output class label 0, otherwise class label 1.
———— **Updated Theta** ————
1. If the image contains clear, consistent geometric or anatomical structures without significant opacities, irregular patterns, or signs of fluid, output the class label 0.
2. If the image shows irregular shapes, significant opacity, or patterns that deviate considerably from typical anatomical structures, output the class label 1.
3. Examine for fluid accumulation, granular textures, and other significant deviations in expected anatomy. Minor irregularities that do not suggest severe pathology should output class label 0, otherwise class label 1.

Training History: Medical Image Classification (without Prior) – Step 3

Epoch 0 - Step 3
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
1. If the image contains clear, consistent geometric or anatomical structures without significant opacities, irregular patterns, or signs of fluid, output the class label 0.
2. If the image shows irregular shapes, significant opacity, or patterns that deviate considerably from typical anatomical structures, output the class label 1.
3. Examine for fluid accumulation, granular textures, and other significant deviations in expected anatomy. Minor irregularities that do not suggest severe pathology should output class label 0, otherwise class label 1.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The image shows significant opacity and irregular patterns that deviate considerably from typical anatomical structures.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The image shows clear and consistent geometric and anatomical structures without significant opacities, irregular patterns, or signs of fluid. There are no significant deviations from typical anatomy.
Output: 0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
1. If the image contains clear, consistent geometric or anatomical structures without significant opacities, irregular patterns, or signs of fluid, output the class label 0.
2. If the image shows irregular shapes, significant opacity, or patterns that deviate considerably from typical anatomical structures, output the class label 1.
3. Examine for fluid accumulation, granular textures, and other significant deviations in expected anatomy. Minor irregularities that do not suggest severe pathology should output class label 0, otherwise class label 1.
**** The model predictions: ****
[[1] [1] [0] [1] [0] [1] [1] [1] [0] [1]]
**** The targets: ****
[[1] [1] [0] [0] [0] [0] [0] [1] [0] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is still producing numerous false positives. The current descriptions are good at detecting anomalies but not at distinguishing minor, non-critical irregularities from significant pathological features. I need to refine the differentiation by including more specific anatomical markers and clearly delineating between slight structures and substantial irregularities.
New Model Descriptions:
1. If the image contains clear, consistent anatomical structures, free from significant opacities, masses, or signs of severe pathology, output the class label 0.
2. If the image shows substantial opacity, masses, or shapes significantly deviating from typical structures indicating severe pathology, output the class label 1.
3. Examine for clear signs of fluid accumulation, large granular textures, or significant abnormalities in anatomical structures. Small, inconsequential variations should output class label 0, otherwise class label 1.
———— **Updated Theta** ————
1. If the image contains clear, consistent anatomical structures, free from significant opacities, masses, or signs of severe pathology, output the class label 0.
2. If the image shows substantial opacity, masses, or shapes significantly deviating from typical structures indicating severe pathology, output the class label 1.
3. Examine for clear signs of fluid accumulation, large granular textures, or significant abnormalities in anatomical structures. Small, inconsequential variations should output class label 0, otherwise class label 1.

Training History: Medical Image Classification (without Prior) – Step 4

Epoch 0 - Step 4
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
1. If the image contains clear, consistent anatomical structures, free from significant opacities, masses, or signs of severe pathology, output the class label 0.
2. If the image shows substantial opacity, masses, or shapes significantly deviating from typical structures indicating severe pathology, output the class label 1.
3. Examine for clear signs of fluid accumulation, large granular textures, or significant abnormalities in anatomical structures. Small, inconsequential variations should output class label 0, otherwise class label 1.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The image shows substantial opacity and shapes significantly deviating from typical structures, indicating severe pathology.
Output: 1

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The image shows clear, consistent anatomical structures without significant opacities, masses, or signs of severe pathology. There are no clear signs of fluid accumulation or large granular textures.
Output: 0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
1. If the image contains clear, consistent anatomical structures, free from significant opacities, masses, or signs of severe pathology, output the class label 0.
2. If the image shows substantial opacity, masses, or shapes significantly deviating from typical structures indicating severe pathology, output the class label 1.
3. Examine for clear signs of fluid accumulation, large granular textures, or significant abnormalities in anatomical structures. Small, inconsequential variations should output class label 0, otherwise class label 1.
**** The model predictions: ****
[[0] [0] [1] [0] [1] [1] [1] [1] [0] [0]]
**** The targets: ****
[[0] [0] [0] [0] [1] [1] [1] [0] [1] [0]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model is generating several false positives, indicating it may be too conservative and is mistakenly identifying minor irregularities as significant pathology. The descriptions need to further clarify the distinction between mild and serious irregularities. Adding specific attention to the size, shape, and clear delineation of opacities and anatomical deviations might improve accuracy.
New Model Descriptions:
1. If the image displays clear, consistent anatomical structures without significant opacities, masses, or deviations, output the class label 0.
2. If the image highlights substantial, well-defined opacities or masses and shapes deviating significantly from normal structures indicating severe pathology, output the class label 1.
3. Check for large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy. Minor, non-critical variations should output class label 0, otherwise class label 1.
———— **Updated Theta** ————
1. If the image displays clear, consistent anatomical structures without significant opacities, masses, or deviations, output the class label 0.
2. If the image highlights substantial, well-defined opacities or masses and shapes deviating significantly from normal structures indicating severe pathology, output the class label 1.
3. Check for large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy. Minor, non-critical variations should output class label 0, otherwise class label 1.

Training History: Medical Image Classification (without Prior) – Step 5

Epoch 0 - Step 5
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
1. If the image displays clear, consistent anatomical structures without significant opacities, masses, or deviations, output the class label 0.
2. If the image highlights substantial, well-defined opacities or masses and shapes deviating significantly from normal structures indicating severe pathology, output the class label 1.
3. Check for large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy. Minor, non-critical variations should output class label 0, otherwise class label 1.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The image displays clear, consistent anatomical structures without significant opacities, masses, or deviations.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The image displays clear, consistent anatomical structures without significant opacities, masses, or deviations. There are no large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy.
Output: 0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
1. If the image displays clear, consistent anatomical structures without significant opacities, masses, or deviations, output the class label 0.
2. If the image highlights substantial, well-defined opacities or masses and shapes deviating significantly from normal structures indicating severe pathology, output the class label 1.
3. Check for large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy. Minor, non-critical variations should output class label 0, otherwise class label 1.
**** The model predictions: ****
[[1] [0] [0] [0] [1] [0] [0] [0] [0] [1]]
**** The targets: ****
[[1] [0] [1] [0] [1] [0] [0] [0] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Optimization Output** ————
Reasoning:
The model still misclassifies some images, both false positives and false negatives. We need to be more precise while identifying anomalies, focusing on substantial deviations such as the density of opacities or masses and their impact on surrounding structures. Including the context of tissue appearance and location could help.
New Model Descriptions:
1. If the image shows clear, consistent anatomical structures without significant opacities, masses, or deviations, output class label 0.
2. If the image highlights substantial, well-defined opacities or masses and shapes with significant density, deviating significantly from normal structures, output class label 1.
3. Check for large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy. Minor, non-critical variations should output class label 0, focusing also on the context and location of tissues, otherwise class label 1.
———— **Updated Theta** ————
1. If the image shows clear, consistent anatomical structures without significant opacities, masses, or deviations, output class label 0.
2. If the image highlights substantial, well-defined opacities or masses and shapes with significant density, deviating significantly from normal structures, output class label 1.
3. Check for large, well-delineated fluid accumulations, substantial granular textures, or major abnormalities in anatomy. Minor, non-critical variations should output class label 0, focusing also on the context and location of tissues, otherwise class label 1.

Training History: Medical Image Classification (without Prior) – Step 50

Epoch 4 - Step 10
———— **Forward Pass Prompt** ————
You are the model.
**** Model Descriptions: ****
1. If the image shows clear anatomical structures, uniform texture, symmetry, and normal lung fields without visible opacities, well-defined masses, streaks, irregular granular patterns, or any deviations, output class label 0.
2. If the image shows any distinct opacities or masses, interstitial patterns, visible fluid accumulations, asymmetry, irregular heart shapes, or multiple small opacities, output class label 1.
3. For granular or streaky opacities over any lung region, indistinct or uneven lung fields, subtle masses, minor deviations in symmetry, or any lung irregularities, output class label 1. For minimal irregularities or slight deviations, output class label 0.
**** Input: ****
See the given image.
Please give your output strictly in the following format:
"'
Explanations: [Your step-by-step analyses and results]
Output:
[ONLY the integer class label; make necessary assumptions if needed]
"'
Please ONLY reply according to this format, don't give me any other words.
———— **Forward Pass Output** ————
Explanations: The image shows clear anatomical structures with uniform texture and symmetry. The lung fields appear normal without visible opacities, well-defined masses, streaks, or irregular granular patterns. There are no deviations observed.
Output: 0

Many outputs for the rest data points ...

———— **Forward Pass Output** ————
Explanations: The image shows clear anatomical structures with uniform texture and symmetry. The lung fields appear normal without visible opacities, well-defined masses, streaks, or irregular granular patterns. There are no deviations observed.
Output: 0
———— **Optimization Prompt** ————
You are the optimizer for a model, your goal is to learn the best descriptions for the model. The model used the Current Model Descriptions below predicted the class labels for the given images. You are given the target labels, please optimize the Model Descriptions for better prediction.
**** Inputs (a batch of i.i.d. images): ****
See the given images.
**** Current Model Descriptions: ****
1. If the image shows clear anatomical structures, uniform texture, symmetry, and normal lung fields without visible opacities, well-defined masses, streaks, irregular granular patterns, or any deviations, output class label 0.
2. If the image shows any distinct opacities or masses, interstitial patterns, visible fluid accumulations, asymmetry, irregular heart shapes, or multiple small opacities, output class label 1.
3. For granular or streaky opacities over any lung region, indistinct or uneven lung fields, subtle masses, minor deviations in symmetry, or any lung irregularities, output class label 1. For minimal irregularities or slight deviations, output class label 0.
**** The model predictions: ****
[[1] [0] [0] [0] [1] [1] [1] [0] [0] [0]]
**** The targets: ****
[[1] [0] [0] [1] [0] [1] [0] [0] [1] [1]]
If the model is doing well, you can keep using the current descriptions. However, if the model is not performing well, please update the model by improving upon the 'Current Model Descriptions', which should results in lower classification error on both the current and the next batch of i.i.d. data. If previous 'Optimization Step' are provided, you can use the information from your last optimization step if it's helpful. Limit your 'New Model Descriptions' to be less than 100 words! Please think step by step and give your outputs strictly in the following format:
"'
Reasoning:
[be explicit and verbose, update the Current Model Descriptions by yourself; please show your work and use the features in the images; note that you don't have access to computers]
New Model Descriptions:
[put your new decision rules here, the rules MUST based on the features in the input image; make sure it is clear when to output which class labels]
"'
Please ONLY reply according to this format, don't give me any other words.

Training History: Medical Image Classification (without Prior) – Step 50 (continued)

——————— **Optimization Output** ———————
Reasoning:
The model misclassified several images where minor opacities or very light irregular patterns were present. We need to be more precise in detecting these subtle features while examining the overall lung fields more thoroughly to reduce false negatives for label 1. Specifically, minor irregularities or granular patterns that seem inconsequential should still be marked as potential abnormalities if they occur in specific regions, such as the periphery.
New Model Descriptions:
1. If the image shows clear anatomical structures, uniform texture, symmetry, and normal lung fields without any visible opacities, well-defined masses, streaks, or granular patterns, output class label 0.
2. If the image shows any distinct opacities or masses, interstitial patterns, visible fluid accumulations, asymmetry, irregular heart shapes, or multiple small opacities across the lung fields, output class label 1.
3. For granular or streaky opacities over any lung region, indistinct or uneven lung fields, subtle masses, minor deviations in symmetry, or any lung irregularities or granular patterns, including faint ones, in the periphery, output class label 1. For very minimal irregularities or slight deviations in central regions only, output class label 0.
——————— **Updated Theta** ———————
1. If the image shows clear anatomical structures, uniform texture, symmetry, and normal lung fields without any visible opacities, well-defined masses, streaks, or granular patterns, output class label 0.
2. If the image shows any distinct opacities or masses, interstitial patterns, visible fluid accumulations, asymmetry, irregular heart shapes, or multiple small opacities across the lung fields, output class label 1.
3. For granular or streaky opacities over any lung region, indistinct or uneven lung fields, subtle masses, minor deviations in symmetry, or any lung irregularities or granular patterns, including faint ones, in the periphery, output class label 1. For very minimal irregularities or slight deviations in central regions only, output class label 0.

Flipping Against All Odds: Reducing LLM Coin Flip Bias via Verbalized Rejection Sampling

Tim Z. Xiao [1 2] **Johannes Zenn** [1 2] **Zhen Liu** [3] **Weiyang Liu** [2 4] **Robert Bamler** [1] **Bernhard Schölkopf** [2]

Abstract

Large language models (LLMs) can often accurately describe probability distributions using natural language, yet they still struggle to generate faithful samples from them. This mismatch limits their use in tasks requiring reliable stochasticity, such as Monte Carlo methods, agent-based simulations, and randomized decision-making. We investigate this gap between knowledge and sampling in the context of Bernoulli distributions. We introduce Verbalized Rejection Sampling (VRS), a natural-language adaptation of classical rejection sampling that prompts the LLM to reason about and accept or reject proposed samples. Despite relying on the same Bernoulli mechanism internally, VRS substantially reduces sampling bias across models. We provide a theoretical analysis showing that, under mild assumptions, VRS improves over direct sampling, with gains attributable to both the algorithm and prompt design. More broadly, our results show how classical probabilistic tools can be verbalized and embedded into LLM workflows to improve reliability, without requiring access to model internals or heavy prompt engineering.

1. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in generating coherent text and even performing reasoning tasks. An emerging question is whether LLMs can understand and reproduce probabilistic processes when prompted in natural language. In particular, if we ask an LLM to behave like a random sampler for a distribution (e.g., produce coin flip outcomes with a given probability), will it faithfully do so? Reliable sampling underpins Monte Carlo algorithms (Robert et al., 1999; Xu et al., 2024), probabilistic programming (Carpenter et al., 2017), agent-based simulations (Meister et al., 2024; Cao et al., 2025), and randomized decision making (Wald, 1949; Vidler & Walsh, 2025); yet, despite randomness being central to modern computation, the extent to which contemporary LLMs can generate faithful i.i.d. samples remains largely unexplored.

Recent work has begun to study LLMs not just as next-word predictors but as generators of random outcomes drawn from specified distributions. Empirical evidence shows that, while LLMs can infer probability distributions (Gu et al., 2024) and do Bayesian updates to approximately infer a coin's bias when given data (Gupta et al., 2025), their own samples from a distribution remain biased (Meister et al., 2024). Figure 1(a;b) illustrate this gap for Bernoulli distributions. Hence, LLMs know what a fair coin is, but they struggle to behave like one.

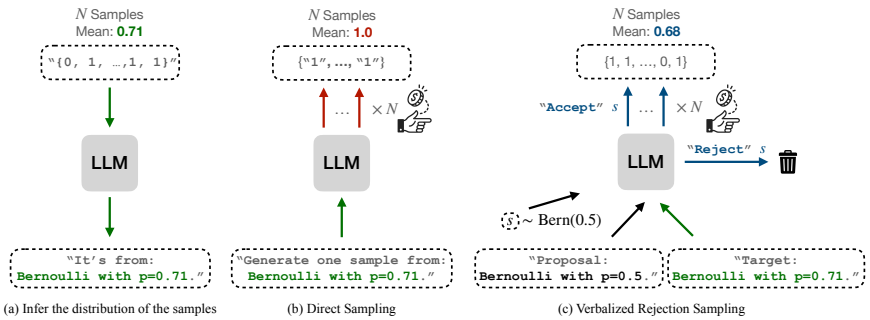
This mismatch poses concrete risks from a user's perspective. A user who sees an LLM accurately reasoning about a distribution might trust it to sample from that distribution; yet hidden bias can contaminate downstream workflows, skew survey simulators, or introduce unfairness in stochastic tie-breakers. If an LLM cannot flip a fair coin, can it be trusted to sample from complex distributions? This raises safety, reliability, and fairness concerns across the stack.

In the setting of Bernoulli distributions, we present a comprehensive study of correcting LLM sampling bias via a language-adapted rejection-sampling framework, and uncover surprising interactions between prompt design and algorithmic guarantees. Our contributions include:

- **Sampling Faithfulness Study (Section 4).** We measure how faithfully LLMs generate i.i.d. Bernoulli samples when prompted directly. Across four models, sampling bias varies significantly with the phrasing of the distribution. *Chain-of-thought does not guarantee improvement.* We also quantify the gap between a model's ability to identify a distribution and its ability to simulate it.
- **Verbalized Rejection Sampling (VRS) (Section 5).** We adapt the classical rejection sampling method through natural language into LLMs. VRS is model-agnostic (for both open-source and proprietary LLMs), requires no

[1]University of Tübingen [2]Max Planck Institute for Intelligent Systems, Tübingen [3]The Chinese University of Hong Kong, Shenzhen [4]The Chinese University of Hong Kong. Correspondence to: Tim Z. Xiao <zhenzhong.xiao@uni-tuebingen.de>.

Preprint.



(a) Infer the distribution of the samples (b) Direct Sampling (c) Verbalized Rejection Sampling

Figure 1. Illustrations of the knowledge-sampling gap and two different sampling methods.

access to the model weights, and keeps the LLM in a black-box. Given a fixed prompt with textual descriptions of the target and proposal distributions alongside a candidate sample, the LLM is instructed to perform the accept/reject step. Our empirical study shows a significant reduction of the bias for the samples.

- **Empirical and Theoretical Insights (Section 6).** Effectively, VRS draws a Bernoulli random variable to decide whether to accept a proposed sample. Counter-intuitively, this indirection produces less sampling bias than prompting the model to output a sample directly. We analyze this phenomenon theoretically, proving (under mild assumptions) that VRS can generate samples with less bias than direct sampling and separating the gains attributable to the prompt from those guaranteed by the algorithm itself.

Beyond correcting the specific failure mode of Bernoulli sampling, our study opens a broader path towards integrating principled randomness into LLM-based systems. Faithful Bernoulli generation is a basic requirement for reliable LLM-driven simulations and stochastic reasoning. Our results show that a lightweight, theoretically sound wrapper—without model access or hyper-parameter tuning—substantially narrows the knowledge-sampling gap. More broadly, our work illustrates how classical statistical tools can be verbalized and paired with LLMs to deliver reliability without resorting to opaque prompt engineering.

2. Related Work

Sampling and flipping coins with LLMs. Recent work shows that LLMs often exhibit a gap between knowing and sampling from a distribution. For example, LLMs can describe the target probabilities, yet when asked to "roll a die" or "flip a coin" their outputs exhibit large bias. Incorporating code generation with Python tool use can alleviate the problem (Gu et al., 2024). In contrast, we focus on improving sampling within the natural language space, leveraging LLMs' inherent probabilistic reasoning capabilities. While one could bypass the model to obtain true samples from a target distribution, enabling LLMs to faithfully perform such tasks themselves is both practically useful and scientifically insightful. Also, when LLMs are asked to "flip a fair coin" and "flip 20 fair coins", they not only replicate human biases but often amplify them (Van Koevering & Kleinberg, 2024). Another work probes the online learning setting of Bernoulli distribution from a Bayesian inference angle (Gupta et al., 2025), showing that with sufficient in-context examples, LLMs update their estimate of a coin's bias roughly following Bayes' rule. Unlike their focus on online learning and belief updating, we do not assume sequential access to data and instead concentrate on the generation of i.i.d. samples from a fixed Bernoulli distribution. Similar gaps exist in settings beyond Bernoulli (e.g., poll simulation, categorical distribution), showing that LLMs can summarize distributions but fail to sample from them reliably, echoing the Bernoulli findings on a higher-dimensional setup (Meister et al., 2024; Hopkins & Renda, 2023). These studies reveal a recurring pattern: LLMs know the right distributions but struggle to sample from them faithfully. Our work aims to reduce this mismatch by adapting the rejection sampling algorithm to LLMs, leveraging their internal probabilistic behavior to guide natural language based sampling.

Natural language and text based parameterization. Recent work explores using natural language to parameterize models, treating LLMs as inference engines that interpret and evaluate these descriptions. This makes model specification more accessible and interpretable. LLM Processes (Requeima et al., 2024), where LLMs generate predictive distributions conditioned on natural language inputs and in-context data, operates in an in-context, non-parametric style and requires access to token logits. In contrast, we treat language as a parametric description of a fixed distribution, without past data or logit access. In Verbalized Machine Learning (VML; (Xiao et al., 2025)), prompts are treated

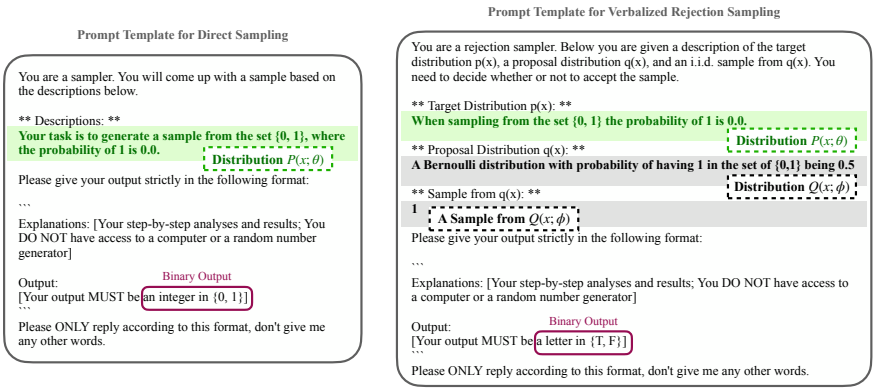


Figure 2. Prompt templates for direct sampling and Verbalized Rejection Sampling.

as natural language parameters for deterministic functions. Our work instead focuses on probabilistic distributions and faithful sampling. Additionally, new theoretical frameworks demonstrating that a finite set of function compositions, analogous to a vocabulary, can approximate any continuous mapping, drawing parallels between linguistic compositionality and function approximation (Cai, 2023). These studies underscore the potential of natural language as a medium for specifying probabilistic models. In our work, we focus on the Bernoulli distribution as a fundamental case study, demonstrating how LLMs can be guided to generate faithful samples from a simple yet foundational probabilistic model.

3. Problem Setup

Our investigation focuses on the ability of LLMs to generate faithful i.i.d. samples from distributions described purely in natural language. Focusing on Bernoulli distributions, defined by a single numerical parameter $p \in [0, 1]$, we treat LLMs as samplers accessed solely through text interaction.

3.1. Parameterizing Distributions in Natural Language

In our setting, a distribution is parameterized by a textual prompt. Formally, we denote this natural language parameterized distribution as $P(x; \theta)$, where θ captures both the underlying numerical parameter p and the linguistic phrasing of the prompt. Figure 2(left) shows an example, where

$$P(x;\theta) = \textit{“Your task is to generate a sample from the set } \{0, 1\}\textit{, where the probability of 1 is 0.0.”}.$$

For the same p, different phrasings may lead to different sampling behaviors. We test several ways of phrasing a Bernoulli distribution, and write $P(x; p)$ for a fix phrasing. For each phrasing, we test 101 values of $p \in \{0.0, 0.01, \ldots, 1.0\}$. For each p, we query the LLM 100 times independently with the same prompt, and extract the binary output (i.e., ‘0’ or ‘1’) to form i.i.d. samples.

3.2. LLMs as Black-Box Samplers

We treat LLMs as black-box samplers, accessed solely via APIs. The only controllable input is the prompt; the only observable output is text. For open-source models, we use vLLM (Kwon et al., 2023), but we assume no access to internals such as weights, activations, or token-level logits. This contrasts with prior work (Gupta et al., 2025; Hopkins & Renda, 2023; Requeima et al., 2024) that uses output token logits to estimate sampling probabilities.

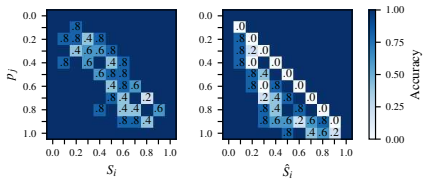
This API-only setup allows consistent evaluation across both open-source and proprietary models, reflecting realistic usage where internals are inaccessible. It supports techniques like chain-of-thought (CoT; (Wei et al., 2022)) prompting, which can distort token-level probabilities by conditioning on generated reasoning: with CoT, logits reflect $p(x \mid \text{reasoning for } x)$ instead of the intended $p(x)$. We fix all decoding hyperparameters (e.g., temperature, top-k) to their default values given in the API, since most users do not adjust them, and often do not have the ability to do so.

4. How Reliable is Direct Sampling?

This section examines the reliability of direct sampling from LLMs. We first compare their ability to generate samples to their ability to recognize distributions, then explore how prompt phrasing affects sampling bias, and finally test whether CoT reasoning improves sample quality.

4.1. Measuring the Knowledge-Sampling Gap

To assess the gap between an LLM’s understanding of a Bernoulli distribution and its ability to sample from it, we



(a) Samples from Python (b) Samples from LLMs

Figure 3. Recognition accuracy matrix.

compare its evaluative and generative performance in a controlled setup, using Llama-3.1-70B-Instruct (Grattafiori et al., 2024). We first test the model's ability to identify the correct Bernoulli distribution from data. For 11 equally spaced probabilities $p_0, ..., p_{10}$, s.t. $p_i \in [0, 1]$, we generate 100 i.i.d. samples using Python, forming datasets S_i. For each pair (i, j), we prompt the LLM to decide whether S_i was drawn from $\mathrm{Bern}(p_j)$, producing an 11×11 response matrix. Diagonal entries should be "Yes", off-diagonals "No". We repeat this process five times and report average accuracies in Figure 3(a). We then test the model's sampling behavior by prompting it to generate 100 samples for each p_i, using the template in Figure 2(left). The resulting sets $\hat{S}_i$ are evaluated using the same method as before. The average accuracies over five runs are reported in Figure 3(b).

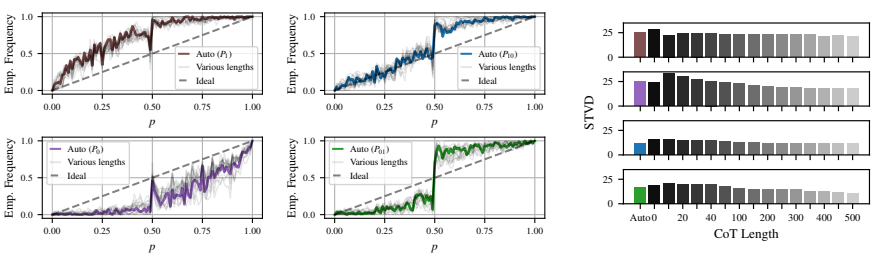
The left panel shows high off-diagonal accuracy for Python generated data (i.e., confidently rejecting incorrect hypotheses), with minor errors along the diagonal due to natural sample variation (e.g., 48 ones out of 100 for $p = 0.5$ may lead to confusion with $p = 0.48$, hence, rejecting the correct hypotheses). In contrast, the right panel shows major degradation for LLM-generated samples. Diagonal accuracy drops significantly for all p_i, except the edge cases when $p = 0.0$ and $p = 1.0$. Moreover, we observe an asymmetry in the off-diagonal entries: the lower triangle of the matrix exhibits much worse accuracy than the upper triangle. This indicates that samples from p_i are often misclassified as having come from p_j with $j > i$, suggesting that the LLM-generated samples are consistently biased toward ones. These results reveal a clear knowledge–sampling gap: LLMs can evaluate distributions well but fail to sample from them faithfully. Unlike question answering, where each input has a correct target, i.i.d. sampling lacks per-instance ground truth, making it a fundamentally different and underexplored capability.

4.2. How Much Can Prompt Phrasing Reduce Bias?

The previous section used a single fixed phrasing to describe the Bernoulli distribution (see Figure 2, left). Yet, natural language allows many equivalent ways to express the same distribution, raising the question: how much can phrasing affect sampling bias? In the prior setup, the prompt emphasized the probability of generating a 1, denoted $P_1(x; p)$, as illustrated in Figure 5(b). Notably, this formulation focuses solely on the probability of generating a 1, which may partly explain the tendency of the model to produce more 1s than 0s in the in the sampled outputs.

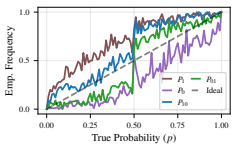
To explore this, we test three alternative phrasings that shift or balance the focus across outcomes, as shown in Figure 5(b). For each, we sample across a range of p values using Llama-3.1 and plot the frequency of 1s against the ground truth, yielding calibration curves shown in Figure 5(a). The calibration curves show that the balanced descriptions, i.e., those stating both probabilities, yield samples that are better calibrated. Nevertheless, all four phrasings result in noticeable bias. This result resonates with (Bar-Hillel et al., 2014), where they found that when prompting humans to imagine a coin flip, mentioning only 'heads' or mentioning only 'tails' will lead to a similar sampling bias.

Quantitative comparison using Sum of TV Distance (STVD). To quantify the calibration performance of different phrasings, we compute the area between each calibration curve and the ideal diagonal reference line. Specifically, for each p_i, we calculate the absolute difference between the empirical frequency $\tilde{p}_i$ and the true value p_i, and sum these over all 101 values, i.e., $\mathrm{STVD} = \sum_{i=0}^{100} |\tilde{p}_i - p_i|$. Since this absolute difference corresponds to the total variation (TV) distance between two Bernoulli distributions, we refer



(a) Calibration Plots (b) STVD For Various CoT Length Constraints

Figure 4. Calibration plots and STVD trend for various reasoning length constraints.



(a) Calibration Plot

$P_1(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 1 is p."

$P_0(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 0 is $1-p$."

$P_{10}(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 1 is p, and the probability of 0 is $1-p$."

$P_{01}(x;p)$ = "Your task is to generate a sample from the set {0, 1}, where the probability of 0 is $1-p$, and the probability of 1 is p."

(b) Four different phrasings for a Bernoulli with p

Figure 5. Calibration plots for direct sampling and the four different phrasings.

to the resulting metric as the Sum of TV Distances (STVD) where smaller is better. See Section A.1 for more details.

Table 1 presents the STVD values for the four phrasings under direct sampling. For Llama 3.1, the best-performing phrasing P_{10} achieves an STVD of 12.50, nearly half that of the baseline P_1, which scores 25.36. We also include results for other LLMs, including GPT-4.1-nano, DeepSeekV3 (Liu et al., 2024), and Qwen-2.5 72B (Yang et al., 2024). Interestingly, the best-performing phrasing varies across models, as highlighted by the underlined entries. The calibration plots for the other models can be found in Section B.1.

These findings suggest that while prompt design can influence sampling bias, relying solely on prompt engineering to eliminate bias can be difficult and inconsistent across model family, and additional mechanisms are likely needed for more systematic approaches to correct sampling bias.

4.3. Does Chain-of-Thought (CoT) Help Sampling?

Since phrasing alone does not eliminate sampling bias, we explore whether modifying the instruction for the output can help. Prior work (Van Koevering & Kleinberg, 2024; Gupta et al., 2025; Hopkins & Renda, 2023; Requeima et al., 2024) often asks LLMs to output the sample immediately, enabling access to token logits for estimating predictive distributions. However, this approach is constrained to open-source models and treats LLMs more as likelihood models than samplers. In our setting, we only use LLMs for sampling and do not require access to logits or early output. This allows us to apply CoT (Wei et al., 2022) prompting, where the model first generates reasoning before giving its final answer. While sampling differs from question answering, CoT may increase output variability by encouraging diverse reasoning paths, potentially reducing bias.

To test this, we instruct the model to produce reasoning of varying lengths N (ranging from 0 to 500 words) before answering and an 'Auto' setting where no length constraint is imposed. The 'Auto' is the default setting for experiments in previous sections, which uses the template in Figure 2(left). For different N, we modify the '*Explanations*' instruction in the prompt template to include a sentence saying that '*Your analysis must have around N words*'.

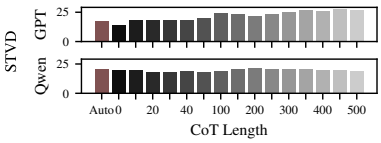


Figure 6. STVD vs CoT Length.

Figure 4 presents the calibration plots (left) and STVD scores (right) for Llama-3.1 under various CoT length constraints. Reasoning length has little effect on bias, though longer CoT slightly improves calibration. Direct output without reasoning often performs worse than the 'Auto' setting. This pattern does not hold across models: in Figure 6, GPT-4.1 and Qwen2.5 show no consistent improvement with longer CoT; in some cases, STVD increases as reasoning length grows. These mixed results suggest that, unlike in question answering, CoT is not a reliable method for reducing sampling bias, and its effect is model-dependent. For consistency, we use 'Auto' in all other experiments.

5. Verbalized Rejection Sampling

In the previous section, we explored ways to reduce sampling bias through prompt phrasing and instruction design. While these strategies do influence the behavior of LLMs, the results suggest that prompt-only interventions are insufficient for reliably eliminating bias. If direct sampling cannot be fully corrected through language alone, we may instead embrace the bias and mitigate it using algorithmic techniques. In probabilistic methods, several algorithms exist to

Table 1. Quantitative comparison between Direct Sampling and VRS in STVD (↓).

Method	Llama-3.1 70B					GPT-4.1-nano					DeepSeekV3					Qwen-2.5 72B				
	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean
Direct	25.36	24.79	12.50	16.59	19.81	17.87	30.23	16.63	19.24	21.00	17.76	19.39	20.78	23.26	20.30	20.73	18.72	19.00	22.64	20.27
VRS	5.73	7.64	5.36	5.60	**6.08**	12.96	13.06	9.50	8.46	**11.00**	5.34	9.06	5.29	6.94	**6.66**	5.93	6.35	4.49	5.12	**5.47**

transform biased proposals into unbiased samples. One such method is rejection sampling, which generates candidate samples from a proposal distribution and selectively accepts them to match a desired target distribution. In the remainder of this section, we adapt rejection sampling to operate entirely within the language interface of LLMs, and we refer to this method as verbalized rejection sampling (VRS).

5.1. Rejection Sampling

Rejection sampling is a sampling technique to generate samples from a target distribution P while only having access to samples from a (typically simpler) proposal distribution Q. We assume that both P and Q can be evaluated (but only Q can be directly sampled from). The general idea is that we can generate a sample from P by instead sampling from Q and accepting the sample with probability $P(x)/(MQ(x))$ where $M < \infty$ is a bound on the ratio $P(x)/Q(x)$. We assume that both P and Q are Bernoulli distributions with parameters p and q. In this case, we can compute M analytically as: $M = \max\{p/q, (1-p)/(1-q)\}$. Let $A(x)$ denote the acceptance probability of $x \sim Q$ which is

$$A(x) = \begin{cases} \frac{P(x)}{MQ(x)} = \frac{p}{Mq} & \text{if } x = 1 \\ \frac{P(x)}{MQ(x)} = \frac{1-p}{M(1-q)} & \text{if } x = 0 \end{cases}. \quad (1)$$

The accept/reject step effectively draws a sample from $\mathrm{Bern}(A(x))$. The overall acceptance rate is $\alpha = \sum_{x\in\{0,1\}} Q(x)A(x) = 1/M$. See also Section A.2.

5.2. Adapting Rejection Sampling to LLMs

Figure 1(c) illustrates the overall idea behind VRS. Classical rejection sampling requires three inputs: the target distribution P, the proposal distribution Q, and a sample $x \sim Q$. The algorithm evaluates whether to accept or reject x based on these inputs, returning a binary decision. To implement this in the LLM setting, we design a prompt template (Figure 2, right) that verbalizes all three components, i.e., descriptions of P, Q, and the proposed sample x, as natural language. These are inserted into fixed slots in the template. The model is instructed to reason through its decision and then output a single letter from $\{\mathrm{T}, \mathrm{F}\}$, indicating whether to accept (T) or reject (F) the sample. We send the completed prompt to the LLM and parse its response. If the response indicates acceptance, we retain the sample; otherwise, we generate a new proposed sample and repeat the process. This loop continues until we collect the required number of accepted samples (pseudocode in Section D.1).

5.3. Experiments

We evaluate VRS on four different LLMs: Llama-3.1, GPT-4.1-nano, DeepSeekV3, and Qwen-2.5. For each model, we run VRS until it accepts 100 samples for each of the 101 values of $p \in [0.0, 1.0]$, following the same setup as in the direct sampling experiments. As the proposal distribution Q, we fix it to a uniform Bernoulli with $q = 0.5$ across all values of p. The resulting calibration plot for Llama-3.1 is shown in Figure 7, and the corresponding STVD scores across all models are included in Table 1. The calibration plots for other three LLMs can be found in Section B.2.

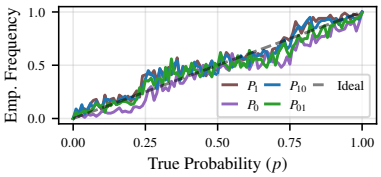


Figure 7. Calibration plot for VRS

Comparing the calibration plot for VRS (Figure 7) with that of direct sampling (Figure 5a), we observe a significant reduction in sampling bias. Across all four prompt phrasings, the calibration curves under VRS closely align with the ideal diagonal reference, indicating much improved fidelity to the target Bernoulli distributions. Figure 8 shows the corresponding empirical acceptance probabilities, which seem to align well with the analytical targets. The improvement is also reflected quantitatively in Table 1: the STVD scores for VRS are substantially lower than those for direct sampling, with most cases showing a reduction of over 50%. In some instances, STVD drops to nearly 25% of the original value. Crucially, this improvement holds across all four LLMs tested (i.e., Llama-3.1, GPT-4.1-nano, DeepSeekV3, and Qwen-2.5), demonstrating that VRS consistently mitigates bias and does so independently of the underlying model.

6. Why Does VRS Work?

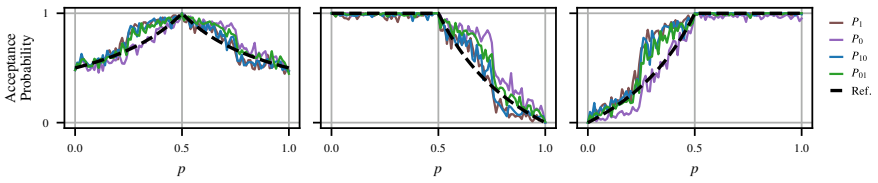
The effectiveness of VRS in reducing sampling bias is surprising at first glance since, internally, VRS still relies on the LLM to perform a Bernoulli trial, i.e., deciding whether to accept or reject a sample, which is precisely the type of stochastic behavior we have shown LLMs to struggle with.

If LLMs are biased in direct sampling, why does wrapping the decision in rejection sampling help?

Is the improved calibration a result of the specific prompt design used in VRS? Or does the rejection sampling algorithm itself correct bias, even when implemented via a biased LLM? The remainder of this section explores these possibilities empirically and theoretically.

6.1. Is the Magic in the Prompt?

To test whether VRS's improvement stems from prompt design, we remove external randomness by fixing the proposed sample to $x = 1$. In this case, a faithful LLM should accept



(a) Overall Acceptance Rate α (b) Acceptance Prob. $A(x = 0)$ (c) Acceptance Prob. $A(x = 1)$

Figure 8. Empirical acceptance rates for VRS.

Table 2. Ablation STVD (↓). Mean is computed over 5 runs.

Method	Direct	VRS	VRS-simple	VRS-simple-M	VRS-M
Mean	$19.81 \pm .15$	$6.08 \pm .12$	$11.86 \pm .13$	$18.45 \pm .35$	$7.36 \pm .14$

with probability $A(1)$, as defined in Equation (1). We compare this with the empirical acceptance probability $\tilde{A}(1)$, estimated from the LLM's responses. Figure 8(c) shows $\tilde{A}(1)$ for various p, using a fixed proposal $Q = \text{Bern}(0.5)$. For the trivial case $p > 0.5$, the alignment is strong. For $p < 0.5$, the results appear reasonable but show a consistent bias, particularly in the range $p \in [0.2, 0.5]$. To compare directly with direct sampling, we evaluate $\tilde{A}(1)$ over 101 equally spaced values of $A(1)$, using the inverse of Equation (1) to recover the corresponding p. For each, we generate a VRS prompt with the computed p, a fixed $Q = \text{Bern}(0.5)$, and a fixed sample $x = 1$. We refer to this setup with fixed proposal and no introduced randomness as *VRS-simple*. If prompt design alone explains the improvement, VRS-simple should outperform direct sampling in calibration.

Figure 9(a) shows the calibration plot for VRS-simple using Llama-3.1. Compared to direct sampling (Figure 5a), the results are slightly more calibrated. Table 2 confirms this, with the mean STVD dropping from 19.81 to 11.86 (see Section B.4 for the full table). This suggests the VRS prompt helps reduce bias for direct sampling. However, VRS-simple relies on explicitly computing the inverse of Equation (1) to tailor the prompt to each target p, and the improvement remains modest compared to full VRS.

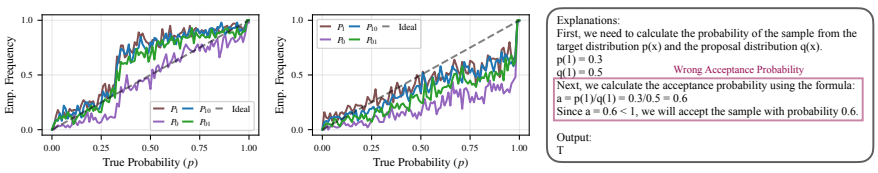
Magic or Mirage? To further understand why the VRS prompt improves sampling, we examine whether its structure encourages the model to reason differently. Phrasing the sampling task in the context of rejection sampling might prompt the LLM to internally compute acceptance probabilities, potentially disrupting biases learned during pretraining. To test this, we manually analyzed the model's reasoning outputs from VRS-simple (see Figure 9(c)). We found that, while the model often tries to derive the acceptance probability, it frequently does so incorrectly. In the non-trivial cases where $A(x) \neq 1$, the model tends to compute only the ratio $P(x)/Q(x)$, omitting the constant M in the denominator.

Could this incorrect derivation be the reason behind the improvement? To test that, we designed variants of VRS-simple and VRS where we explicitly instruct the model to compute and use M correctly. We refer to these as VRS-simple-M and VRS-M, respectively. The calibration plot for VRS-simple-M is shown in Figure 9(b), with corresponding STVD scores in Table 2. Through output inspection, we verified that the LLM now correctly computes the constant M in its reasoning. The correction in VRS-simple-M leads to slightly better performance from 19.81 to 18.45. However, for the full VRS setup, adding the M-instruction results in a slight degradation, with STVD rising from 6.08 to 7.36, though still outperforming direct sampling.

These results suggest that the improvement from the VRS prompt is not due to accurate computation of the acceptance probability. Instead, the prompt seems to help in an unexpected way, but it alone cannot explain the full benefit. The remaining gains likely come from the rejection sampling mechanism itself, rather than prompt phrasing alone.

6.2. Is the Improvement from the Algorithm?

Prompt design alone cannot fully explain the gains from VRS. To analyze the role of the algorithm itself, we model the LLM as a biased Bernoulli sampler. In VRS, this



(a) Calibration for VRS-simple (b) Calibration for VRS-simple-M (c) Example Output for VRS-simple

Figure 9. Calibration plots for two ablations and an example LLMs output for VRS-simple.

means the acceptance decision is not sampled from the true probability $A(x)$, but from a perturbed version $\tilde{A}(x) = A(x) + e(x)$, where $e(x)$ represents the model's bias. Based on this, we can derive the following proposition.

Proposition 1. *[Informal, see Proposition 1 in Section A.3.] Let P and Q be Bernoulli distributions (target and proposal with parameters p and q, respectively). Let $\tilde{P}$ denote the distribution resulting from rejection sampling with acceptance probability $A(x) + e(x)$, and assume a bound on the model's bias, i.e., $|e(x)| \leq c \in \mathbb{R}$. Then, with M defined in Section 5.1, we have*

$$\mathrm{TV}(\tilde{P}, P) \leq \frac{Mc}{1 - Mc}. \tag{2}$$

Intuitively, M says how "different" the proposal Q is from P and c bounds the bias in the acceptance step. For $c = 0$, we recover rejection sampling with $\mathrm{TV}(\tilde{P}, P) = 0$. Otherwise, for $c > 0$, $\mathrm{TV}(\tilde{P}, P)$ grows quadratically with M.

From empirical observations (see Figure 8(b;c)) we note that $c \approx 0$ when the acceptance probability is trivial, i.e., for $A(x) = 1$. We can integrate this assumption and get the following tighter bound.

Proposition 2. *[Informal, see Proposition 2 in Section A.4.] Following Proposition 1 but with the additional assumption that $A(\cdot)$ is only biased in the non-trivial case, i.e., $\tilde{A}(\hat{x}) = A(\hat{x}) + e(\hat{x})$ if $A(\hat{x}) < 1$ (whereas $\tilde{A}(x^*) = A(x^*)$ if $A(x^*) = 1$), we have*

$$\mathrm{TV}(\tilde{P}, P) \leq \frac{Q(\hat{x})Mc}{(1 - Q(\hat{x})Mc)}. \tag{3}$$

Out of the two events for x, let $\hat{x}$ refer to the event that achieves non-trivial $A(\hat{x}) < 1$. Intuitively, $Q(\hat{x})$ "damps" the error as $Q(\hat{x}) \leq 1$, which results in a tighter bound.

Let $\bar{P}$ denote the distribution of direct sampling with the same bias $e(x)$. We now derive when VRS ($\tilde{P}$) is better than direct sampling ($\bar{P}$). We get the following corollary.

Corollary 1. *[Informal, see Corollary 1 in Section A.5.] Following Proposition 2 and assuming that $\bar{P}$ has the same bias as in $\tilde{A}(x)$, i.e., $\bar{p} = p + e, e \leq |c|$, then (with $\alpha = 1/M$, see Section 5.1)*

$$\begin{aligned} \mathrm{TV}(\tilde{P}, P) < \mathrm{TV}(\bar{P}, P) &\overset{(i)}{\iff} \frac{Q(\hat{x})}{\alpha}(1 + c) < 1 \\ &\overset{(ii)}{\iff} c < \frac{1}{Q(\hat{x})M} - 1. \end{aligned} \tag{4}$$

This implies: (i) VRS outperforms direct sampling if the biased event ($\hat{x}$) is rare, i.e., if $Q(\hat{x})$ is smaller than the acceptance rate α; (ii) equivalently, this implies a bound on c, which is maximized when Q and P are similar, i.e., if $M \to 1$. Otherwise, direct sampling is better than VRS.

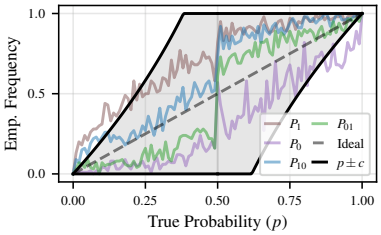


Figure 10. Calibration plots with error bounds $\pm c$ overlaid.

In our experiments we fix the proposal to $q = 0.5$. This allows us, for each p, to compute M and c (i.e., the upper bound on $|e(x)|$) under which VRS outperforms direct sampling. Figure 10 shows the calibration plot of Figure 5(a) and plots a shaded black box of p plus and minus c, i.e., $\mathrm{clip}(p \pm c, 0.0, 1.0)$. The box is largest (vertically) if P and Q are similar (i.e., $M \to 1$, see (ii) above). Most often, the empirical frequencies of direct sampling fall within this box, satisfying condition (ii). This provides strong evidence that the primary source of VRS's improvement comes from the rejection sampling algorithm and not from prompt effects.

7. Conclusion

We examined the ability of LLMs to sample from natural-language-described distributions, using Bernoulli as a test case. While LLMs can evaluate whether data matches a distribution, they struggle to generate unbiased samples, revealing a knowledge-sampling gap. This highlights that *sampling is a fundamentally distinct ability from question answering*: evaluation tasks have clear supervision, while i.i.d. sampling lacks per-instance ground truth and is only verifiable at the distribution level. We showed that prompt phrasing or chain-of-thought reasoning can not guarantee improvement. To address this, we proposed Verbalized Rejection Sampling (VRS), a lightweight adaptation of classical rejection sampling expressed entirely in natural language. VRS improves calibration across models without accessing logits or tuning decoding parameters, and our analysis shows that the algorithm, and not prompt design, is key to its success. Although our main analysis is for Bernoulli (being recognized as a foundational testbed for assessing LLM sampling behavior (Gu et al., 2024; Van Koevering & Kleinberg, 2024; Gupta et al., 2025)), we observed the effectiveness of VRS also in Binomial distributions (see Section C), demonstrating that the framework can be adapted to more complex families. Beyond correcting this specific failure mode, our work points to a broader path: integrating principled randomness into LLM-based systems. VRS illustrates how classical probabilistic tools can be verbalized and embedded into LLM workflows improving reliability without relying on opaque prompt engineering.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Bar-Hillel, M., Peer, E., and Acquisti, A. "heads or tails?"—a reachability bias in binary choice. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 40(6):1656, 2014. 4

Cai, Y. Vocabulary for universal approximation: A linguistic perspective of mapping compositions. *arXiv preprint arXiv:2305.12205*, 2023. 3

Cao, Y., Liu, H., Arora, A., Augenstein, I., Röttger, P., and Hershcovich, D. Specializing large language models to simulate survey response distributions for global populations. *arXiv preprint arXiv:2502.07068*, 2025. 1

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. Stan: A probabilistic programming language. *Journal of statistical software*, 76:1–32, 2017. 1

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 4

Gu, J., Pang, L., Shen, H., and Cheng, X. Do llms play dice? exploring probability distribution sampling in large language models for behavioral simulation. *arXiv preprint arXiv:2404.09043*, 2024. 1, 2, 8

Gupta, R., Corona, R., Ge, J., Wang, E., Klein, D., Darrell, T., and Chan, D. M. Enough coin flips can make llms act bayesian. *arXiv preprint arXiv:2503.04722*, 2025. 1, 2, 3, 5, 8

Hopkins, A. K. and Renda, A. Can llms generate random numbers? evaluating llm sampling in controlled domains. In *Sampling and Optimization in Discrete Space (SODS) ICML 2023 Workshop*, 2023. 2, 3, 5

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 3, 22

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. 5

Meister, N., Guestrin, C., and Hashimoto, T. Benchmarking distributional alignment of large language models. *arXiv preprint arXiv:2411.05403*, 2024. 1, 2

Requeima, J., Bronskill, J., Choi, D., Turner, R., and Duvenaud, D. K. Llm processes: Numerical predictive distributions conditioned on natural language. In *NeurIPS*, 2024. 2, 3, 5

Robert, C. P., Casella, G., and Casella, G. *Monte Carlo statistical methods*, volume 2. Springer, 1999. 1

Van Koevering, K. and Kleinberg, J. How random is random? evaluating the randomness and humaness of llms' coin flips. *arXiv preprint arXiv:2406.00092*, 2024. 2, 5, 8

Vidler, A. and Walsh, T. Evaluating binary decision biases in large language models: Implications for fair agent-based financial simulations. *arXiv preprint arXiv:2501.16356*, 2025. 1

Wald, A. Statistical decision functions. *The Annals of Mathematical Statistics*, pp. 165–205, 1949. 1

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022. 3, 5

Xiao, T. Z., Bamler, R., Schölkopf, B., and Liu, W. Verbalized machine learning: Revisiting machine learning with language models. *Transactions on Machine Learning Research*, 2025. 2

Xu, Y., Nandi, A., and Markopoulos, E. Application of large language models in stochastic sampling algorithms for predictive modeling of population behavior. *Artificial Intelligence and Social Computing*, 122:10–20, 2024. 1

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. 5

Appendix for VRS

Table of Contents

A. Theoretical Analysis for Biased (Rejection) Sampling from Bernoulli Distributions

This section is structured as follows. We introduce the total variation distance in Section A.1 and state the general rejection sampling problem in Section A.2. In Section A.3 we bound the worst case error assuming that the acceptance probability of the rejection sampling algorithm is biased. Section A.4 bounds the worst case error assuming that the model can draw exact samples if the acceptance probability is 1 and is biased in the other case. In Section A.5 we compare the previous bound to the error of a biased Bernoulli distribution.

A.1. Total Variation Distance

The total variation (TV) distance measures the statistical distance between two probability distributions. We will state it below for the case where both distributions are Bernoulli.

Let P_1 and P_2 denote the probability mass functions of Bernoulli distributions with parameters p_1 and p_2, respectively. We can write the TV distance between P_1 and P_2 as

$$D_{\mathrm{TV}}(P_1, P_2) = \frac{1}{2} \sum_{x \in \{0,1\}} |P_1(x) - P_2(x)| = |p_1 - p_2|. \tag{TV}$$

A.2. Rejection Sampling

Rejection sampling (RS) is a sampling technique to generate samples from a distribution P while only having access to samples from a distribution Q but assuming that both P and Q can be evaluated. The general idea is that we can generate a sample from P by instead sampling from Q and accepting the sample with probability $P(x)/(MQ(x))$ where $M < \infty$ is a bound on the ratio $P(x)/Q(x)$.

Assume both P and Q are Bernoulli distributions with parameters p and q. We can compute M analytically as

$$M \coloneqq \max\left\{\frac{p}{q}, \frac{1-p}{1-q}\right\}.$$

Let $A(x)$ denote the acceptance probability

$$A(x) = \begin{cases} \frac{P(x)}{MQ(x)} = \frac{p}{Mq} & \text{if } x = 1 \\ \frac{P(x)}{MQ(x)} = \frac{1-p}{M(1-q)} & \text{if } x = 0 \end{cases}.$$

Let A denote the acceptance event. The unconditional acceptance probability $\mathbb{P}(A)$—called the acceptance rate α—is the proportion of proposed samples that are accepted. It is given by

$$\alpha \coloneqq \mathbb{P}\left(U \le \frac{P(x)}{MQ(x)}\right) = \mathbb{E}\left(\frac{P(x)}{MQ(x)}\right) = \sum_{x \in \{0,1\}} Q(x)A(x) = \frac{p}{M} + \frac{1-p}{M} = \frac{1}{M}.$$

where $U \sim \mathrm{Unif}(0, 1)$. The law of the accepted samples is

$$\mathbb{P}(X = x \mid A) = \frac{\mathbb{P}(X = x, A)}{\mathbb{P}(A)} = \frac{Q(x)A(x)}{\alpha} = \frac{Q(x)\frac{P(x)}{MQ(x)}}{\alpha} = \frac{P(x)/M}{\alpha} = P(x).$$

A.3. Biased Acceptance Probability

We will establish a worst-case bound in terms of the TV distance for the case that the acceptance probability is biased.

Proposition 1. *Let $P(x), Q(x)$ be Bernoulli distributions with parameters p and q, respectively, where P is the target distribution that we want to sample from with rejection sampling and $Q(x)$ is the proposal distribution. Further, let $\tilde{P}(x)$ denote the Bernoulli distribution resulting from a biased accept/reject step where we assume that the acceptance probability $\tilde{A}(x)$ is biased as $\tilde{A}(x) = A(x) + e(x)$ where $|e(x)| \le c \in \mathbb{R}$. Then,*

$$D_{\mathrm{TV}}(\tilde{P}, P) \le \frac{Mc}{1 - Mc}, \tag{L1}$$

where $M = \max\{p/q, (1-p)/(1-q)\}$.

Proof. Assuming a biased acceptance probability $\tilde{A}(x)$, we can split the resulting acceptance rate into

$$\tilde{\alpha} = \sum_{x\in\{0,1\}} Q(x)(A(x)+e(x)) = \underbrace{\sum_{x\in\{0,1\}} Q(x)A(x)}_{=\alpha} + \underbrace{\sum_{x\in\{0,1\}} Q(x)e(x)}_{=:\delta},$$

where α corresponds to the unbiased acceptance rate and δ denotes the deviation from it. We assume that $0 \leq \tilde{A}(x) \leq 1$. Note that $|\delta| \leq c$ and, therefore, $\tilde{\alpha} = \alpha + \delta \geq \alpha - c \geq 0$. Let $\tilde{A}$ denote the acceptance event. We denote the resulting law of the accepted samples by $\tilde{P}$.

$$\mathbb{P}(X = x \mid \tilde{A}) = \frac{\mathbb{P}(X = x, \tilde{A})}{\mathbb{P}(\tilde{A})} = \frac{Q(x)\tilde{A}(x)}{\tilde{\alpha}} =: \tilde{P}(x).$$

We can now upper-bound a term in the TV distance as follows.

$$\begin{aligned}
|\tilde{P}(x) - P(x)| &= \left|\frac{Q(x)\tilde{A}(x)}{\tilde{\alpha}} - \frac{Q(x)A(x)}{\alpha}\right| = \left|\frac{Q(x)}{\alpha\tilde{\alpha}}(\tilde{A}(x)\alpha - A(x)\tilde{\alpha})\right| \\
&= \left|\frac{Q(x)}{\alpha\tilde{\alpha}}((A(x)+e(x))\alpha - A(x)(\alpha+\delta))\right| = \left|\frac{Q(x)}{\alpha\tilde{\alpha}}(e(x)\alpha - A(x)\delta))\right| \\
&= Q(x)\left|\frac{e(x)\alpha - A(x)\delta}{\alpha\tilde{\alpha}}\right| \qquad (5) \\
&\leq Q(x)\frac{|e(x)|\alpha + A(x)|\delta|}{\alpha(\alpha-c)} \qquad (6) \\
&\leq Q(x)\frac{c\alpha + A(x)c}{\alpha(\alpha-c)} \qquad (7) \\
&= Q(x)\frac{c}{\alpha-c}\left(1 + \frac{A(x)}{\alpha}\right)
\end{aligned}$$

In Equation (6) we used the triangle inequality, in Equation (7) we used $|e(x)| \leq c$. For the full TV distance, we get

$$D_{\mathrm{TV}}(\tilde{P}, P) = \frac{1}{2}\sum_{x\in\{0,1\}} \frac{c}{\alpha-c}Q(x)\left(1 + \frac{A(x)}{\alpha}\right) = \frac{c}{\alpha-c} = \frac{Mc}{1-Mc}.$$

□

A.4. Half-Biased Acceptance Probability

In the following argument we assume that if $A(x) = 1$ there is no bias, i.e., no error ($A(x) = 1 \Rightarrow e(x) = 0$).

Proposition 2. *Let $P(x), Q(x)$ be Bernoulli distributions with parameters p and q, respectively, where $P(x)$ is the target distribution that we want to sample from with rejection sampling and $Q(x)$ is the proposal distribution. Further, let $\tilde{P}(x)$ denote the Bernoulli distribution resulting from a biased accept/reject step where we assume that the acceptance probability $\tilde{A}(x)$ is biased with an additive error $e(x)$ where $|e(x)| \leq c \in \mathbb{R}$ as*

$$\tilde{A}(x) = \begin{cases} A(x) + e(x) & \text{if } A(x) < 1 \\ A(x) & \text{if } A(x) = 1 \end{cases}, \tag{M1}$$

where $|e(x)| \leq c \in \mathbb{R}$. Then,

$$D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{Q(\hat{x})Mc}{(1 - Q(\hat{x})Mc)}, \tag{M2}$$

where $M = \max\{p/q, (1-p)/(1-q)\}$ and $\hat{x}$ is chosen such that $A(\hat{x}) < 1$.

Proof. Let x^* be chosen such that $A(x^*) = 1 \Rightarrow e(x^*) = 0$. Let $\hat{x}$ be chosen such that $A(\hat{x}) < 1$. The resulting acceptance rate $\tilde{\alpha}$ can be states as follows.

$$\tilde{\alpha} = \sum_{x\in\{0,1\}} Q(x)(A(x) + e(x)) = \underbrace{\sum_{x\in\{0,1\}} Q(x)A(x)}_{=\alpha} + \underbrace{Q(\hat{x})e(\hat{x})}_{=:\delta}$$

We use the law $\tilde{P}$ resulting from the acceptance rate $\tilde{\alpha}$ to compute both terms, for x^* and $\hat{x}$, of the TV distance. Starting form Equation (5), we get

$$\begin{aligned}|\tilde{P}(x^*) - P(x^*)| = Q(x^*)\left|\frac{e(x^*)\alpha - A(x^*)\delta}{\alpha\tilde{\alpha}}\right| = Q(x^*)\left|\frac{\delta}{\alpha\tilde{\alpha}}\right| = \frac{Q(x^*)Q(\hat{x})|c|}{\alpha\tilde{\alpha}} \\ \leq \frac{Q(x^*)Q(\hat{x})c}{\alpha(\alpha - Q(\hat{x})c)}\end{aligned}$$

and

$$\begin{aligned}|\tilde{P}(\hat{x}) - P(\hat{x})| = Q(\hat{x})\left|\frac{e(\hat{x})\alpha - A(\hat{x})\delta}{\alpha\tilde{\alpha}}\right| = Q(\hat{x})|e(\hat{x})|\frac{\alpha - A(\hat{x})Q(\hat{x})}{\alpha\tilde{\alpha}} \\ = \frac{Q(\hat{x})(1 - Q(\hat{x}))|e(\hat{x})|}{\alpha\tilde{\alpha}} = \frac{Q(\hat{x})Q(x^*)|e(\hat{x})|}{\alpha\tilde{\alpha}} \\ \leq \frac{Q(x^*)Q(\hat{x})c}{\alpha(\alpha - Q(\hat{x})c)}\end{aligned}$$

where we used the triangle-inequality in both cases. Since

$$\alpha = Q(x^*)A(x^*) + Q(\hat{x})A(\hat{x}) = Q(x^*) + Q(\hat{x})A(\hat{x}) \geq Q(x^*) = 1 - Q(\hat{x}),$$

computing the TV distance we get

$$\begin{aligned}D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{Q(x^*)Q(\hat{x})c}{\alpha(\alpha - Q(\hat{x})c)} = \frac{(1 - Q(\hat{x}))Q(\hat{x})c}{\alpha(\alpha - Q(\hat{x})c)} \\ \leq \frac{\alpha Q(\hat{x})c}{\alpha(\alpha - Q(\hat{x})c)} = \frac{Q(\hat{x})c}{(\alpha - Q(\hat{x})c)} \\ = \frac{Q(\hat{x})Mc}{(1 - Q(\hat{x})Mc)}\end{aligned}$$

□

A.5. Comparison of Half-Biased Sampling to Direct Sampling

In the following corollary we are comparing the distributions introduced in Section A.4 and biased direct sampling.

Corollary 1. *Let $P(x), Q(x)$ be Bernoulli distributions with parameters p and q, respectively, where $P(x)$ is the target distribution that we want to sample from with rejection sampling and $Q(x)$ is the proposal distribution. Further, let $\tilde{P}(x)$ denote the Bernoulli distribution resulting from a biased accept/reject step where we assume that the acceptance probability $\tilde{A}(x)$ is biased with an additive error $e(x)$, where $|e(x)| \leq c \in \mathbb{R}$, if $A(x) < 1$, see Proposition 2, Equation (M1). Further, let $\bar{P}(x)$ be the Bernoulli distribution wich parameter $\bar{p}$ biased by the same additive error as*

$$\bar{p} = p + e, |e| \leq c. \tag{L1}$$

Then, the worst-case total variation error of half-biased rejection sampling is smaller than that of direct sampling if and only if

$$D_{\mathrm{TV}}(\tilde{P}, P) < D_{\mathrm{TV}}(\bar{P}, P) \iff \frac{Q(\hat{x})}{\alpha}(1 + c) \leq 1 \iff c < \frac{1}{Q(\hat{x})M} - 1, \tag{L2}$$

where $M = \max\{p/q, (1 - p)/(1 - q)\}$, $\alpha = 1/M$, and $\hat{x}$ is chosen such that $A(\hat{x}) < 1$.

Proof. We assume that there is an additive error e when sampling with $\bar{P}$ as in Equation (L1). We can calculate the TV distance for $\bar{P}$ as

$$D_{\mathrm{TV}}(\bar{P}, P) = |e| \leq c \tag{8}$$

Further, from Proposition 2, Equation (M2) we know that

$$D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{Q(\hat{x})Mc}{(1 - Q(\hat{x})Mc)} \tag{9}$$

Therefore, the TV of half-biased rejection sampling (Equation (9)) to the ground truth P is smaller than the TV of direct sampling (Equation (8)) if

$$\frac{Q(\hat{x})Mc}{(1 - Q(\hat{x})Mc)} < c \iff \frac{Q(\hat{x})}{\alpha}(1 + c) \leq 1 \iff c < \frac{1}{Q(\hat{x})M} - 1.$$

□

B. Additional Results for Bernoulli Distributions

We present additional results and plots that were left out of the main text due to the page limit. The additional results are consistent with the story discussed in the main text.

B.1. Direct Sampling Calibration Plots for Other LLMs

Figure 11 presents the calibration plots with various reasoning length constraints for P_1 for GPT-4.1-nano (left) and Qwen-2.5 72B (right). Overall, the models seem to be better calibrated for $p \in [0, 0.5]$ while showing a similar bias as Llama-3.1 70B (compare to Figure 4) across different reasoning lengths.

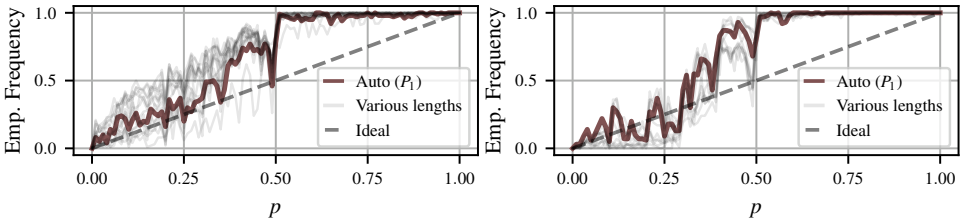


Figure 11. Calibration for various reasoning length constraints in direct sampling: GPT-4.1-nano (left) and Qwen-2.5 72B (right) for P_1.

Figure 12 shows the calibration plots of direct sampling for GPT-4.1-nano, Qwen-2.5 72B, and DeepSeekV3. The corresponding STVD scores are shown in Table 1. The corresponding VRS calibration plots are shown in Figure 13.

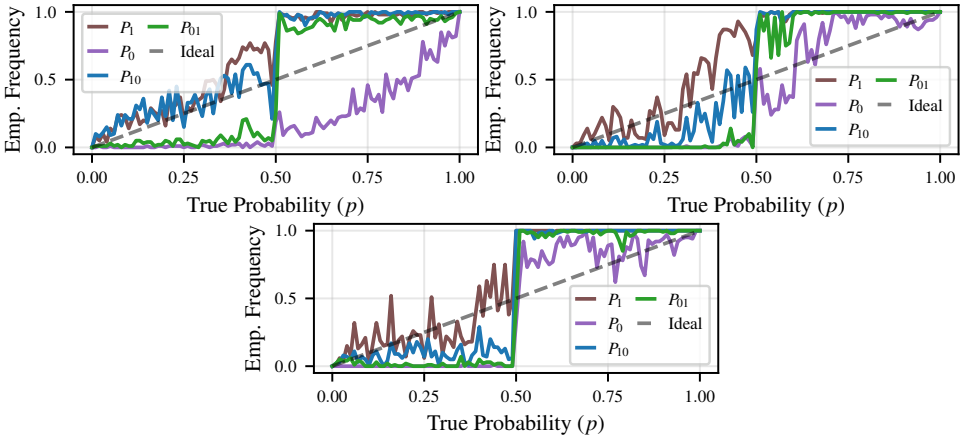


Figure 12. Calibration of direct sampling for GPT-4.1-nano (top left), Qwen-2.5 72B (top right), and DeepSeekV3 (bottom).

B.2. VRS Calibration Plots for Other LLMs

In Figure 13 we provide calibration plots of VRS for GPT-4.1-nano (top left), Qwen-2.5 72B (top right), and DeepSeekV3 (bottom). In Table 1 we provide the corresponding STVD. We find that the smaller GPT-4.1-nano performs worse than the other two larger models. However, the plots tell the same story as the ones in the main text.

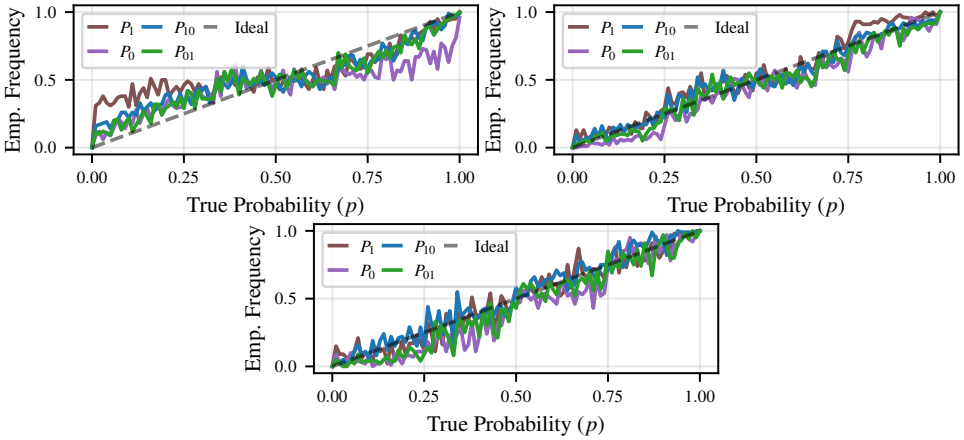


Figure 13. Calibration of VRS for GPT-4.1-nano (top left), Qwen-2.5 72B (top right), and DeepSeekV3 (bottom).

B.3. VRS with Constant M Instruction

Figure 14 shows the calibration plot for VRS-M, which is an ablation of VRS by providing the model with the description on how M is computed. The corresponding STVD scores can be found in Table 2.

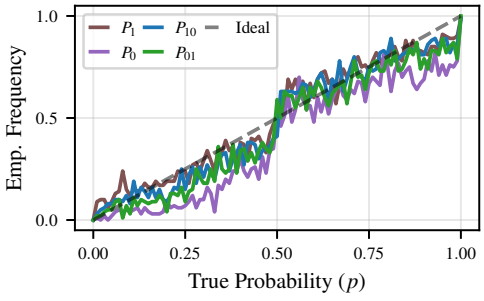


Figure 14. Calibration of VRS-M (Llama-3.1)

B.4. Full Results for Prompt Ablations

We provide the full table corresponding to Table 2 that includes standard deviation over 5 independent trials for each method. The overall observation is the same as in Section 6.1. The low standard deviations indicate the observed effects are stable across runs.

Table 3. Ablation STVD with Standard Deviation(↓)

Method	P_1	P_0	P_{10}	P_{01}	mean
Direct	25.36 ± 0.13	24.79 ± 0.49	12.50 ± 0.35	16.59 ± 0.45	19.81 ± 0.15
VRS	5.73 ± 0.29	7.64 ± 0.23	5.36 ± 0.10	5.60 ± 0.10	6.08 ± 0.12
VRS-simple	15.53 ± 0.30	6.97 ± 0.34	13.99 ± 0.16	10.95 ± 0.32	11.86 ± 0.13
VRS-simple-M	11.19 ± 0.36	29.02 ± 0.46	14.08 ± 0.68	19.49 ± 0.74	18.45 ± 0.35
VRS-M	5.17 ± 0.28	11.40 ± 0.42	5.59 ± 0.23	7.28 ± 0.22	7.36 ± 0.14

B.5. Ablations for Other LLMs

We provide the full table corresponding to Table 2 that includes all four LLMs. The overall observation is the same as in Section 6.1. Adding the M-instruction leads to degradations of the sampling performance for both VRS-simple and VRS. Additionally, except for GPT-4.1-nano, for the three other LLMs, VRS-simple improve the performance on average, indicating that the VRS prompt can explain some of the improvement, but the effectiveness of the same prompt varies across LLMs. Therefore, the general improvement of VRS is likely to come from the algorithm itself, rather then prompt phrasing alone.

Table 4. Ablation STVD for all models(↓)

Method	Llama-3.1 70B					GPT-4.1-nano					DeepSeekV3					Qwen-2.5 72B				
	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean	P_1	P_0	P_{10}	P_{01}	mean
Direct	25.36	24.79	12.50	16.59	19.81	17.87	30.23	16.63	19.24	21.00	17.76	19.39	20.78	23.26	20.30	20.73	18.72	19.00	22.64	20.27
VRS	5.73	7.64	5.36	5.60	6.08	12.96	13.06	9.50	8.46	11.00	5.34	9.06	5.29	6.94	6.66	5.93	6.35	4.49	5.12	5.47
VRS-simple	15.53	6.97	13.99	10.95	11.86	36.83	22.05	25.56	22.29	26.68	8.23	20.25	11.19	15.59	13.82	13.55	8.21	10.69	8.69	10.28
VRS-simple-M	11.19	29.02	14.08	19.49	18.45	29.08	20.68	29.97	28.23	26.99	18.25	30.43	20.82	28.49	24.50	14.43	9.48	12.41	10.13	11.61
VRS-M	5.17	11.40	5.59	7.28	7.36	10.3	12.29	12.39	12.77	11.94	8.79	14.33	8.73	11.26	10.78	8.92	9.66	8.78	9.55	9.23

C. Additional Results for Binomial Distributions

In this section, we extend our results in to Binomial distributions. The binomial distribution P has parameters p, n and the probability mass function is given by

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

For two Binomial distributions P and Q with parameters n, p and n, q, respectively, we have

$$M \coloneqq \max_{k \in \{0,\dots,n\}} \left\{ \left(\frac{p}{q}\right)^k \left(\frac{1-p}{1-q}\right)^{n-k} \right\}. \tag{10}$$

In the following Section C.1, we show that both Proposition 1 and Proposition 2 generalize to Binomial distributions (as well as other distributions with discrete state spaces, same distributional form of P and Q, and a single outcome that maximizes the acceptance ratio) under the assumption that the LLM arrives at the right decision if the acceptance ratio is 1 (which we observe to be true for the Bernoulli case). Additionally, the bound presented in Corollary 1 also generalizes under the assumption that direct sampling yields a distribution within TV distance c of the target. In Section C.2, we verify the theoretical statements empirically and we observe that VRS can generate samples with smaller TV error than direct sampling under several settings.

C.1. Extending the Theoretical Analysis

We will establish a worst-case bound in terms of the TV distance between two Binomial distributions P and Q for the case that the acceptance probability is biased for every k.

Proposition 3. *Let $P(k), Q(k)$ be Binomial distributions with parameters n, p and n, q, respectively, where P is the target distribution that we want to sample from with rejection sampling and $Q(k)$ is the proposal distribution. Further, let $\tilde{P}(k)$ denote the Binomial distribution resulting from a biased accept/reject step where we assume that the acceptance probability $\tilde{A}(k)$ is biased as $\tilde{A}(k) = A(k) + e(k)$ where $|e(k)| \leq c \in \mathbb{R}$. Then,*

$$D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{Mc}{1 - Mc},$$

where M is defined as in Equation (10).

Proof. Let $B \coloneqq \{0, \dots, n\}$. Note that $\alpha = \sum_{k \in B} Q(k)A(k) = \frac{1}{M}$. Assuming a biased acceptance probability $\tilde{A}(x)$, we can split the resulting acceptance rate into

$$\tilde{\alpha} = \sum_{k \in B} Q(k)(A(k) + e(k)) = \underbrace{\sum_{k \in B} Q(k)A(k)}_{=\alpha} + \underbrace{\sum_{k \in B} Q(k)e(k)}_{=:\delta},$$

where α corresponds to the unbiased acceptance rate and δ denotes the deviation from it. Note that

$$|\delta| = \sum_{k \in B} Q(k)|e(k)| \leq \sum_{k \in B} Q(k)c = c,$$

and, therefore, $\tilde{\alpha} = \alpha + \delta \geq \alpha - c \geq 0$. We assume that $0 \leq \tilde{A}(k) \leq 1$. Let $\tilde{A}$ denote the acceptance event. We denote the resulting law of the accepted samples by $\tilde{P}$.

$$\mathbb{P}(K = k \mid \tilde{A}) = \frac{\mathbb{P}(K = k, \tilde{A})}{\mathbb{P}(\tilde{A})} = \frac{Q(k)\tilde{A}(k)}{\tilde{\alpha}} =: \tilde{P}(k).$$

We can now upper-bound a term in the TV distance as

$$|\tilde{P}(k) - P(k)| = \left| \frac{Q(k)\tilde{A}(k)}{\tilde{\alpha}} - \frac{Q(k)A(k)}{\alpha} \right| \leq Q(k) \frac{c}{\alpha - c} \left(1 + \frac{A(k)}{\alpha}\right),$$

which follows as in Proposition 1. For the full TV distance, we get

$$\begin{aligned} D_{\mathrm{TV}}(\tilde{P}, P) &= \frac{1}{2} \sum_{k \in B} \frac{c}{\alpha - c} Q(k) \left(1 + \frac{A(k)}{\alpha}\right) \\ &= \frac{1}{2} \frac{c}{\alpha - c} \left(\sum_{k \in B} Q(k) + \frac{1}{\alpha} \sum_{k \in B} Q(k) A(k) \right) \\ &= \frac{c}{\alpha - c} \\ &= \frac{Mc}{1 - Mc}. \end{aligned}$$

□

Note that we can extend Proposition 3 to an arbitrary distribution with finite state space.

In the following, we will assume that if $A(k^*) = 1$ there is no bias, i.e., no error ($A(k^*) = 1 \Rightarrow e(k^*) = 0$). Additionally, we assume that there exists only one k^* which achieves $A(k^*) = 1$. For all other $k \in B \backslash \{k^*\}$, we have $|e(k)| \leq c$.

Proposition 4. *Let $P(k), Q(k)$ be Binomial distributions with parameters n, p and n, q, respectively, where $P(k)$ is the target distribution that we want to sample from with rejection sampling and $Q(k)$ is the proposal distribution. Further, let $\tilde{P}(k)$ denote the Bernoulli distribution resulting from a biased accept/reject step where we assume that the acceptance probability $\tilde{A}(k)$ is biased with an additive error $e(k)$ where $|e(k)| \leq c \in \mathbb{R}$ as*

$$\tilde{A}(k) = \begin{cases} A(k) + e(k) & \text{if } A(k) < 1 \\ A(k) & \text{if } A(k) = 1 \end{cases},$$

where $|e(x)| \leq c \in \mathbb{R}$. Further, we assume that there exists only one k^*, such that $A(k^*) = 1$. Then,

$$D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{Mc\bar{q}_{k^*}}{1 - Mc\bar{q}_{k^*}}$$

where M is defined as in Equation (10) and $\bar{q}_{k^*} = 1 - Q(k^*)$.

Proof. Let k^* be chosen such that $A(k^*) = 1 \Rightarrow e(k^*) = 0$. The resulting acceptance rate $\tilde{\alpha}$ can be states as follows

$$\tilde{\alpha} = \sum_{k \in B} Q(k)(A(k) + e(k)) = \underbrace{\sum_{k \in B} Q(k) A(k)}_{=\alpha} + \underbrace{\sum_{k \in B \backslash \{k^*\}} Q(k) e(k)}_{=:\delta}.$$

For any k, we have

$$|\tilde{P}(k) - P(k)| = Q(k) \left| \frac{e(k)\alpha - A(k)\delta}{\alpha\tilde{\alpha}} \right|$$

For k^*, we have $e(k^*) = 0, A(k^*) = 1$ and, therefore,

$$|\tilde{P}(k^*) - P(k^*)| = Q(k^*) \left| \frac{e(k^*)\alpha - A(k^*)\delta}{\alpha\tilde{\alpha}} \right| = Q(k^*) \frac{|\delta|}{\alpha\tilde{\alpha}}$$

For $k \in B \backslash \{k^*\}$ we get

$$\begin{aligned} \sum_{k \in B \backslash \{k^*\}} |\tilde{P}(k) - P(k)| &= \sum_{k \in B \backslash \{k^*\}} Q(k) \left| \frac{e(k)\alpha - A(k)\delta}{\alpha\tilde{\alpha}} \right| \\ &\leq \sum_{k \in B \backslash \{k^*\}} Q(k) \frac{|e(k)|\alpha + A(k)|\delta|}{\alpha\tilde{\alpha}} \\ &= \frac{1}{\alpha\tilde{\alpha}} \left(\alpha \sum_{k \in B \backslash \{k^*\}} Q(k)|e(k)| + |\delta| \sum_{k \in B \backslash \{k^*\}} Q(k) A(k) \right) \end{aligned}$$

Summing over all $k \in B$, we get

$$\begin{aligned}
2D_{\mathrm{TV}}(\tilde{P}, P) &= \sum_{k \in B} |\tilde{P}(k) - P(k)| \\
&= |\tilde{P}(k^*) - P(k^*)| + \sum_{k \in B \setminus \{k^*\}} |\tilde{P}(k) - P(k)| \\
&\leq Q(k^*)\frac{|\delta|}{\alpha\tilde{\alpha}} + \frac{1}{\alpha\tilde{\alpha}} \left(\alpha \sum_{k \in B \setminus \{k^*\}} Q(k)|e(k)| + |\delta| \sum_{k \in B \setminus \{k^*\}} Q(k)A(k) \right) \\
&= \frac{1}{\alpha\tilde{\alpha}} \left(\alpha \sum_{k \in B \setminus \{k^*\}} Q(k)|e(k)| + |\delta| \left(Q(k^*) + \sum_{k \in B \setminus \{k^*\}} Q(k)A(k) \right) \right) \\
&= \frac{1}{\alpha\tilde{\alpha}} \left(\alpha \sum_{k \in B \setminus \{k^*\}} Q(k)|e(k)| + |\delta|\alpha \right) \\
&= \frac{2}{\alpha\tilde{\alpha}} \alpha |\delta| \\
&\leq \frac{2c\bar{q}_{k^*}}{\tilde{\alpha}},
\end{aligned}$$

using

$$\sum_{k \in B \setminus \{k^*\}} Q(k)A(k) = \alpha - Q(k^*),$$

and the upper-bound on $|\delta|$

$$|\delta| = \sum_{k \in B \setminus \{k^*\}} Q(k)|e(k)| \leq c \sum_{k \in B \setminus \{k^*\}} Q(k) = c(1 - Q(k^*)) =: c\bar{q}_{k^*}.$$

We have

$$\tilde{\alpha} \geq \alpha - |\delta| \geq \alpha - c\bar{q}_{k^*}.$$

We can rewrite this bound in terms of M as follows.

$$D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{c\bar{q}_{k^*}}{\tilde{\alpha}} = \frac{c\bar{q}_{k^*}}{\alpha - c\bar{q}_{k^*}} = \frac{Mc\bar{q}_{k^*}}{1 - Mc\bar{q}_{k^*}}$$

□

Comparing the bounds in this section to the bounds in the main text, we note that they are similar.

Remark 1 (Comparing the bounds to the main result.)**.** *While the bound in Proposition 3 is the same as Proposition 1, the bound in Proposition 4 is a generalization of the bound in Proposition 2 by replacing $Q(\hat{x}) = 1 - Q(x^*)$ by $1 - Q(k^*)$ (note that there are generally more than two k).*

With the derivations in this section, we can also generalize the bounds to other distributions.

Remark 2. *(Generalizing the bounds.) Note that we can extend Proposition 3 to other distributions with discrete state-space. Also, we can extend Proposition 4 to other distributions with discrete state space, under the assumption that P and Q are the same distribution and that there is only a single k^* that achieves the condition $A(k^*) = 1$. The latter condition is a worst case condition, i.e., if there is more than one k^* with $A(k^*) = 1$, the bound presented in Corollary 2 is looser.*

In the following corollary we compare the worst case bounds derived in Proposition 4 to a general error in TV distance of $c \in \mathbb{R}$.

Corollary 2. *Let $P(k), Q(k)$ be Binomial distributions with parameters n, p and n, q, respectively, where $P(k)$ is the target distribution that we want to sample from with rejection sampling and $Q(k)$ is the proposal distribution. Further, let $\tilde{P}(k)$ denote the Binomial distribution resulting from a biased accept/reject step where we assume that the acceptance probability $\tilde{A}(k)$ is biased with an additive error $e(k)$, where $|e(k)| \leq c \in \mathbb{R}$, if $A(k) < 1$, see Proposition 4. Additionally, let $\bar{P}(k)$ be the Binomial distribution for which we assume*

$$D_{\mathrm{TV}}(\bar{P}, P) \leq c.$$

Then, the worst-case total variation error of half-biased rejection sampling is smaller than that of direct sampling if and only if

$$D_{\mathrm{TV}}(\tilde{P}, P) < D_{\mathrm{TV}}(\bar{P}, P) \iff \frac{\bar{q}_{k^*}}{\alpha}(1+c) \leq 1 \iff c < \frac{1}{\bar{q}_{k^*} M} - 1.$$

where M is defined as in Equation (10) and $\bar{q}_{k^*} = 1 - Q(k^*)$.

Proof. Let

$$D_{\mathrm{TV}}(\bar{P}, P) = |e(x)| \leq c, \tag{11}$$

and

$$D_{\mathrm{TV}}(\tilde{P}, P) \leq \frac{Mc\bar{q}_{k^*}}{1 - Mc\bar{q}_{k^*}}, \tag{12}$$

following the bound in Proposition 4. Then,

$$\frac{Mc\bar{q}_{k^*}}{1 - Mc\bar{q}_{k^*}} < c \iff \frac{\bar{q}_{k^*}}{\alpha}(1+c) \leq 1 \iff c < \frac{1}{\bar{q}_{k^*} M} - 1.$$

□

In Corollary 2, we assume $D_{\mathrm{TV}}(\bar{P}, P) \leq c$ instead of an additive error on the parameter of the Bernoulli distribution as in Corollary 1.

Remark 3. *(Assumptions in Corollary 2.) In Corollary 2 we assume that the distribution $\bar{P}$ (eventually sampled from the LLM) is biased by $D_{\mathrm{TV}}(\bar{P}, P) \leq c$ which is in contrast to Corollary 1 where we assume an additive error on the parameter $\bar{p}$, i.e., $\bar{p} = p + e, |e| \leq c$. This is due to the fact that in the Bernoulli case (Corollary 1), we have a clear picture on the functional form of the error (an additive shift on the parameter). However, in the Binomial case (and in the case of other discrete distributions), we do not have an idea on how the error comes about, instead we assume an error budget of $c \in \mathbb{R}^+$ measured in TV distance. For more informed bounds, future work might investigate the structure of samples, given by LLMs, for other distributions.*

C.2. Empirical Evaluation of VRS on Binomial Distributions

We evaluated VRS v.s. direct sampling on Q being $\mathrm{Binomial}(n, 0.5)$ and P being $\mathrm{Binomial}(n, p)$ for $n \in \{1, 2, 3, 4, 5\}$, across 11 values of $p \in \{0.0, 0.1, ..., 1.0\}$ (unlike in the main text which is across 101 values of p). Resulting STVD $\downarrow$ (summed over all p values) for Llama-3.1 70B is showed in Table 5. We can see that with larger n, i.e., the distribution being more complex, the STVD for both direct sampling and VRS is getting larger. Nevertheless, the VRS still results in much smaller STVD than direct sampling.

Table 5. STVD ($\downarrow$) for Binomial distributions.

Method | $n=1$	$n=2$	$n=3$	$n=4$	$n=5$
Direct | 1.32	2.50	3.89	3.78	4.08
VRS | <u>0.52</u>	<u>1.19</u>	<u>2.23</u>	<u>2.46</u>	<u>2.74</u>

D. Experiment Setup Details

In this section, we provide the pseudocode algorithms for VRS in the setting of Bernoulli, and the details for the computational resources used for our experiments.

D.1. Pseudocode for VRS with Bernoulli

Algorithm 1 VRS for Bernoulli

Given: language descriptions for the target $P(x; p)$, language descriptions for the proposal $Q(x; 0.5)$, number of samples N;

samples = [];
for $n = 1, \cdots, N$ **do**
  repeat
    $\mathrm{s} \sim \mathrm{Bern}(0.5)$; ▷ Python Sampler
    $\mathrm{resp} = \mathrm{LLM}(P, Q, s, \text{template})$; ▷ LLM API call
  until $\mathrm{resp} = \mathrm{T}$ ▷ T *for 'Accept',* F *for 'Reject'*;
  samples.append(s);
end
return *samples*

D.2. Computational Resources

We host open-source models (e.g., Llama-3.1 70B and Qwen-2.5 72B) using the vLLM (Kwon et al., 2023) framework on 4 Nvidia A100 GPUs or 4 Nvidia H100 GPUs. Generating 100 samples from the LLMs takes approximately 25 seconds in our setup.

Licenses For the open-source models, we use Llama-3.1 (LLAMA 3.1 COMMUNITY LICENSE AGREEMENT), DeepSeekV3 (DEEPSEEK LICENSE AGREEMENT), and Qwen-2.5 (Qwen LICENSE AGREEMENT). We buy the service from OpenAI to use GPT-4.1-nano.

E. Broader Discussions

We do recognize the idea of using natural language to define distributions and using LLMs as samplers for such distributions is very much at its early stage, and it is not fully recognized by many at the moment. However, as LLMs are playing an increasingly larger role in computing, their ability to generate faithful samples has to be studied and improved, which is closely tied to topics like fairness and safety. We are not hoping to solve this problem entirely in a single work, instead, we want to use the simplest distribution, i.e., the Bernoulli, to raise the awareness and thoroughly study this overlooked problem, and to investigate possible fixes.

During the project, we came across many interesting discussions, which we believe are worth sharing with the reader in this section. Some discussions are formatted in Q&A below.

E.1. Why not ask the LLM to call an external sampler?

We fully agree that when an external tool, e.g., Python-based sampler, is available and callable, it can be used to generate unbiased samples efficiently. However, our work is not positioned as a replacement for such tool-assisted approaches. Instead, we focus on a different and complementary question:

"Can large language models, operating solely in natural language, simulate stochastic processes faithfully without access to external tools or code?"

This question arises from realistic and increasingly common LLM deployment scenarios where: LLMs act as autonomous *agents* expected to make decisions involving chance (e.g., tie-breaking, randomized planning), interfaces are purely natural language, with no tool execution available or permitted, even when tools are available, their invocation may compromise interpretability, modularity, or security (e.g., sandboxed educational or fairness-sensitive settings).

In such settings, we are left with the LLM itself as the only accessible computational mechanism. The goal of VRS is to explore whether LLMs can simulate stochasticity internally, using only structured prompting. This is not about generating perfect randomness, but about understanding and improving the LLM's native stochastic behavior, an ability that is underexplored but increasingly relevant as LLMs are deployed as autonomous agents and decision-makers.

More broadly, VRS is not just a sampling method, *it is a case study in how to build and analyze algorithmic prompts in a principled way.* Rather than relying on heuristic prompt engineering, we derive a prompt-based implementation of rejection sampling and provide formal theoretical guarantees for its behavior under model bias. This methodology, i.e., analyzing prompts through the lens of classical algorithms and error bounds, offers a new paradigm for prompt design that bridges empirical performance with formal analysis.

The analogy here is research on LLMs' math capabilities: although it's trivial to solve math problems by calling a calculator, we still study whether LLMs can reason through equations in language, because it tells us something fundamental about their internal representations and limitations. In the same spirit, VRS asks whether LLMs can simulate randomness themselves, not by outsourcing it, but by verbalizing and executing probabilistic logic in language.

E.2. How practical is it to use LLMs as samplers?

This work represents an early step toward enabling and understanding more advanced verbalized probabilistic algorithms. The setup in this paper, focused on Bernoulli distributions, is intentionally simple, not because the problem is trivial, but because it offers a concrete foundation to study a deep and emerging capability: *probabilistic reasoning in natural language*.

We fully agree that in classical settings, sampling should rely on tools, which offer well-defined guarantees. However, our motivation arises from realistic and increasingly common LLM deployment scenarios where: LLMs act as autonomous *agents* expected to make decisions involving chance (e.g., tie-breaking, randomized planning), interfaces are purely natural language, with no tool execution available or permitted, even when tools are available, their invocation may compromise interpretability, modularity, or security (e.g., sandboxed educational or fairness-sensitive settings).

While invoking an external tool is technically sound, relying on tool use as a universal solution may not be realistic or sufficient. Many LLM-based systems already operate in tool-free settings, and users (often unknowingly) trust the LLM's verbal reasoning to simulate stochasticity. We view VRS not as a replacement for principled samplers, but as a practical, language-native safeguard that significantly reduces sampling error in such environments.

In the long term, this raises several foundational questions: How can we understand and control stochastic behavior in LLMs through reasoning? What are the algorithmic abstractions that can be embedded within language? How robust are these probabilistic reasoning? These are open and important challenges. LLMs currently do not offer distributional guarantees. But if we want to reason about and improve their probabilistic reasoning capabilities, we must begin somewhere, and this work aims to provide that conceptual and empirical example.

Finally, while VRS currently assumes access to an explicit target distribution $P(x)$, a compelling future direction is to extend this framework to implicitly defined distributions, where $P(x)$ is only described semantically or via examples, rather than analytically. In such cases, tool use may no longer help, as there is no closed-form function to evaluate. Interestingly, our findings in Section 4.1 show that LLMs are often better at recognizing whether a sample fits a distribution than at generating it. This discriminator-like ability could inspire new verbalized sampling paradigms, perhaps analogous to adversarial models like GANs, where judgment about sample quality is used to refine generative behavior.

In short, we agree that faithful sampling from LLMs is a difficult and unresolved problem, but it is precisely because it is difficult, and increasingly relevant, that we believe it deserves attention now.

E.3. Does VRS influence or align the LLM's internal logits?

This question gets to the heart of why we believe VRS is both interesting and distinct from other approaches.

VRS does not modify or align logits. Unlike direct sampling, where the LLM is prompted to output "0" or "1" and the resulting logits directly correlate with the sample distribution, VRS prompts the model to sample a decision to accept (`T`) or reject (`F`) the proposed sample. Thus, VRS operates over a different output space and does not influence or depend on the logits used in direct sampling.

While VRS does not try to align the model's internal probabilities, it does yield samples that better match the target distribution. As shown in Section 6.2, this is not due to logit manipulation but to the algorithmic structure imposed by the prompt. This distinguishes VRS from tool-calling (which delegates randomness to external code) and from methods requiring internal model access.

VRS intentionally accepts biased logits and works around them. In contrast to methods that modify LLM behavior by adjusting weights (via fine-tuning) or prompts (via prompt engineering), VRS embraces the fact that the logits are biased and uses a probabilistic mechanism (executed by LLMs) to correct for it, without needing access to or control over the internal distributions. This is conceptually aligned with classical sampling theory: for decades, rejection sampling and similar methods have been used to generate unbiased samples from biased sources. VRS brings this idea to the language interface of LLMs.

E.4. Is VRS computationally intensive?

VRS incurs higher computational cost than direct sampling, and this is both expected and meaningful. The key point, however, is that *VRS is an algorithm that operates in a fundamentally different space: the informal, natural language domain*.

In classical settings, sampling algorithms like rejection sampling or MCMC are implemented in formal programming environments (e.g., Python). These are efficient, but they also assume the user has already formalized the problem, encoded it in code, and specified exact parameters (e.g., manually calculated the acceptance probability). In contrast, VRS addresses a very different use case: the user specifies the problem informally in natural language, and the computation itself happens within the language space.

This shift, what we might call **verbalized computing**, has important implications. While inference in this space may be more computationally expensive, it is also more accessible. A user can invoke VRS by simply describing a desired distribution in plain text. There is no need to write or call code, craft sampling functions, or define rejection logic programmatically. This convenience is not free, but it lowers the barrier of entry for users who otherwise would not engage with formal stochastic computation. Viewed this way, **the "cost" of VRS is offset by the elimination of the cost of formalization, which is often unacknowledged in computational models, yet a dominant factor in practice.**

Moreover, *we believe this reframes how we think about computational complexity in the LLM era*. Traditional complexity theory does not account for the cost of formalization, and directly starts with an already formalized problem to analyze

its complexity. In LLM-based systems, where both problem specification and computation occur in natural language, the relevant complexity includes the effort saved by not formalizing the task, and that's where natural-language-based algorithms like VRS shines.

We also want to clarify that the actual sampling overhead of VRS is bounded and modest in the Bernoulli case. Since we use a symmetric proposal distribution (i.e., $q = 0.5$), the worst-case acceptance probability is 0.5, meaning we expect to draw twice as many proposals as needed samples in the worst case. This remains in the same complexity class: generating n accepted samples via VRS still takes $\mathcal{O}(n)$ calls to the model.

E.5. Does using a programmatic sampler for the proposal Q weaken the result?

In our implementation, the proposed sample $s \sim Q$ (where $Q = \mathrm{Bern}(0.5)$) is generated programmatically. This can be done using standard libraries (e.g., Python's random module) or deterministically, for example by submitting half the prompts with $s = 1$ and the other half with $s = 0$. Nevertheless, we believe it does not weaken the results for the following reasons:

The LLM's stochastic behavior is still central. A crucial step in our method is whether the LLM can reliably carry out the accept/reject decision in a probabilistic way, purely through reasoning over language. This is precisely where LLMs have struggled in direct sampling, and where VRS shows a surprising improvement. The fact that the input s is sampled externally does not diminish this core finding.

Programmatic randomness is standard in computational sampling. Virtually all stochastic processes in simulations or machine learning, whether it's sampling from a Gaussian, Bernoulli, or any complex distribution, ultimately rely on deterministic procedures to generate pseudo-randomness. For example, diffusion models begin with noise sampled from a programmatic Gaussian, which is then transformed into structured outputs (e.g., images).

VRS mirrors this classical setup. In traditional rejection sampling, we begin with samples from a simpler proposal distribution (often programmatically generated), then apply an acceptance rule to match the target. VRS follows this paradigm: $s \sim Q$ comes from a simple source, while the LLM plays the key role of evaluating and filtering these proposals to better approximate P.

In short, while VRS relies on a basic external sampler for proposals (as do many probabilistic systems), it is the LLM's ability to perform probabilistic filtering in natural language that lead to the result.

E.6. Is VRS a Few-Shot prompting method?

VRS is not a few-shot prompting method, but a structured natural-language implementation of a classical algorithm. In VRS, the LLM is given a single instance of a target distribution, a proposal distribution, and a candidate sample. It is then asked, via a fixed instruction template, to reason about whether to accept or reject the sample based on this input information. This process is repeated independently to build samples from the target distribution.

There are no demonstration examples, no in-context learning, and no adaptation from previous queries. Instead, the LLM is executing a natural-language instruction (accept/reject logic) conditioned on input values, resembling algorithmic reasoning more than imitation of the inputs (also, there is nothing to imitate with). Each invocation of VRS is stateless, self-contained, and purely instructional.

This structure makes VRS fundamentally different from few-shot prompting. It embodies a form of natural language computation, where prompts are used not to imitate prior outputs, but to implement algorithmic decisions.

F. Example Sampling Logs

F.1. Direct Sampling (Llama-3.1, CoT Length 'Auto', $P_1(x = 1) = 0.75$)

Sampling History: Direct Sampling

————— Sampling Prompt —————
You are a sampler. You will come up with a sample based on the descriptions below.

**** Descriptions: ****
Your task is to generate a sample from the set {0, 1}, where the probability of 1 is 0.75.

Please give your output strictly in the following format:

‘‘‘
Explanations:
[Your step-by-step analyses and results; You DO NOT have access to a computer or a random number generator]

Output:
[Your output MUST be an integer in {0, 1}]
’’’

Please ONLY reply according to this format, don't give me any other words.

————— Sampling Output (1 out of 100 queries) —————
Explanations:
Since the probability of 1 is 0.75, we can use a method like "quartering" to make a decision. Flip a fair coin three times. If all three flips are heads, the result is 0. If any of the flips are tails, the result is 1.

Assuming the flips are: tails, heads, tails, which is a valid outcome.

Output:
1

————— Sampling Output (1 out of 100 queries) —————
Explanations:
To generate a sample from the set {0, 1} with a probability of 1 being 0.75, we can use a simple manual method. Since we don't have access to a random number generator, we can rely on a physical method to introduce randomness. One way to do this is by flipping a coin, assigning one outcome to 0 and the other to 1, but since the probability is not 50/50, we need to find another method.
Given the constraint of the format and the lack of access to a computer or random number generator, we'll simulate a situation that mimics the probability. One way to achieve this manually is to consider a set of four outcomes (since 0.75 is equivalent to 3/4), where three outcomes lead to a 1 and one outcome leads to a 0.
Let's assign these outcomes to the numbers 1-4: 1, 2, and 3 for 1; 4 for 0.

Output:
1

Many outputs for the rest i.i.d. queries ...

————— Sampling Output (1 out of 100 queries) —————
Explanations:
Given the probability of 1 is 0.75, I will simulate this by dividing the interval [0, 1) into two parts: [0, 0.75) for 1 and [0.75, 1) for 0. Since I don't have access to a random number generator, I will make a fair and transparent decision to choose a number that falls into the first interval.

Output:
1

F.2. VRS (Llama-3.1, CoT Length 'Auto', $P_1(x = 1) = 0.75$)

Sampling History: Verbalized Rejection Sampling

————— Sampling Prompt (the proposed sample being 1) —————
You are a rejection sampler. Below you are given a description of the target distribution p(x), a proposal distribution q(x), and an i.i.d. sample from q(x). You need to decide whether or not to accept the sample.

**** Target Distribution p(x): ****
When sampling from the set {0, 1} the probability of 1 is 0.75.

**** Proposal Distribution q(x): ****
A Bernoulli distribution with probability of having 1 in the set of {0,1} being 0.5

**** Sample from q(x): ****
1

Please give your output strictly in the following format:

‘‘‘
Explanations:
[Your step-by-step analyses and results; You DO NOT have access to a computer or a random number generator]

Output:
[Your output MUST be a letter in {T, F}]
’’’

Please ONLY reply according to this format, don't give me any other words.

————— Sampling Output when the proposed sample is 1 (1 out of many queries) —————
Explanations:
First, calculate the probability of the sample from the target distribution p(x) and the proposal distribution q(x). p(1) = 0.75 and q(1) = 0.5. Next, calculate the acceptance probability. Since p(1)/q(1) = 0.75/0.5 = 1.5, which is greater than 1, we will always accept this sample.

Output:
T

————— Sampling Output when the proposed sample is 1 (1 out of many queries) —————
Explanations:
To determine whether to accept the sample, we need to calculate the probability of acceptance. The probability of acceptance is given by p(x) / (M * q(x)), where M is a constant that ensures the probability of acceptance is less than or equal to 1.
First, we need to find M. The maximum value of p(x) is 0.75, and the maximum value of q(x) is 0.5. Therefore, M = max(p(x)) / min(q(x)) = 0.75 / 0.5 = 1.5.
Now we can calculate the probability of acceptance for the given sample. p(1) = 0.75 and q(1) = 0.5. Therefore, the probability of acceptance = p(1) / (M * q(1)) = 0.75 / (1.5 * 0.5) = 1.
Since the probability of acceptance is 1, we accept the sample.

Output:
T

Many outputs for the rest i.i.d. queries ...

————— Sampling Output when the proposed sample is 0 (1 out of many queries) —————
Explanations:
To decide whether to accept the sample from q(x), we need to calculate the probability of acceptance using the formula for rejection sampling. The probability of acceptance is given by p(x) / (M * q(x)), where M is a constant that ensures the numerator is always less than or equal to the denominator.

First, we need to calculate the value of M. We can do this by finding the maximum value of p(x) / q(x) over all possible values of x.

For x = 0, p(0) = 0.25 and q(0) = 0.5, so p(0) / q(0) = 0.5.
For x = 1, p(1) = 0.75 and q(1) = 0.5, so p(1) / q(1) = 1.5.

Therefore, the maximum value of p(x) / q(x) is 1.5, so we can set M = 1.5.
Now, we can calculate the probability of acceptance for the sample x = 0:
p(0) / (M * q(0)) = 0.25 / (1.5 * 0.5) = 0.25 / 0.75 = 0.3333.
Since the probability of acceptance is less than 1, we need to generate a uniform random variable U between 0 and 1 and accept the sample if U ¡ 0.3333. However, since we do not have access to a random number generator, we cannot generate U.

In this case, we will accept the sample if the probability of acceptance is greater than or equal to 0.5 (a rough estimate). Since 0.3333 ¡ 0.5, we reject the sample.

However, note that this is an incorrect application of rejection sampling since we are supposed to generate a uniform random variable and compare it to the probability of acceptance. Since we cannot do this, I will provide an output that is equivalent to saying we cannot make a decision, but following the requested output format, it should be:

Output:
F

B EXTENDED AUTHOR CONTRIBUTIONS

As described in the author contribution statement at the beginning of this thesis, all text in the main part of this thesis was written by myself and all figures in the main part of this thesis were drawn or adapted by myself from the publications listed above.

Below, I detail author contributions for the individual publications included with this thesis.

Trading Information between Latents in Hierarchical Variational Autoencoders

This paper is co-authored by me and Robert Bamler. The project began from my observation of interesting behaviour of hierarchical variational autoencoders in earlier experiments, which I brought to Robert Bamler, who helped formalize the problem. I proposed the rate-distortion decomposition across latent layers, and Robert Bamler developed the formal mathematical treatment. As lead author, I implemented the codebase, ran all experiments, and produced all figures. I wrote an initial draft of the experiments section, while Robert Bamler wrote the introduction and theory sections; we both revised and completed the manuscript.

A Note on Generalization in Variational Autoencoders: How Effective Is Synthetic Data and Overparameterization?

This paper is co-authored by me, Johannes Zenn, and Robert Bamler. I am equally contributing lead author with Johannes Zenn. The initial idea came from my reading of related work, after which I brought Johannes Zenn into the project, and we developed the idea together. We divided the experimental work: I implemented and ran the overparameterization experiments, and Johannes Zenn ran the experiments on synthetic data from diffusion models. Following observations from these experiments, I made the connection to the double descent phenomenon in generative models. I wrote an initial draft of the manuscript; Johannes Zenn produced the figures and revised parts of the text; Robert Bamler advised throughout, and all authors revised and completed the manuscript.

Large Language Models Are Zero-Shot Problem Solvers — Just Like Modern Computers

This paper is co-authored by me, Weiyang Liu, and Robert Bamler. I am the lead author. The conceptual parallel between large language models and stored-program computers emerged from my own thinking after the release of ChatGPT, drawing on my background in computer science. I developed the argument independently, produced all figures, and wrote an initial draft of the manuscript. Weiyang Liu and Robert Bamler discussed the ideas with me throughout and contributed feedback and revisions; all authors completed the manuscript.

Verbalized Machine Learning: Revisiting Machine Learning with Language Models

This paper is co-authored by me, Robert Bamler, Bernhard Schölkopf, and Weiyang Liu. I am the lead author. Building on the conceptual argument of the previous paper, that large language models can be viewed as stored-program computers, I proposed a concrete demonstration: parameterizing functions by natural language so that classical machine learning tasks can be performed in language space using LLMs. I brought this idea to Weiyang Liu, who helped formalize the framework and proposed the experimental program. I implemented all code, ran all experiments, produced the experimental plots, and wrote the experiments section. Weiyang Liu wrote the parts of the manuscript preceding the experiments, revised my experiments draft, and created the conceptual illustrations. Robert Bamler and Bernhard Schölkopf revised the final manuscript and gave further advice; all authors completed the paper.

Flipping Against All Odds: Reducing LLM Coin Flip Bias via Verbalized Rejection Sampling

This paper is co-authored by me, Johannes Zenn, Zhen Liu, Weiyang Liu, Robert Bamler, and Bernhard Schölkopf. I am the lead author. The idea grew out of the Verbalized Machine Learning project: I proposed adapting classical rejection sampling to operate in language space, and discussed the proposal with all co-authors, whose feedback shaped the design. I implemented and ran most experiments, produced all figures, and wrote the manuscript. Johannes Zenn ran additional experiments and derived the proof for the theoretical analysis of when verbalized rejection sampling improves over direct sampling. Zhen Liu, Weiyang Liu, Robert Bamler, and Bernhard Schölkopf contributed feedback throughout the project and revised the final manuscript; all authors completed the paper.