Verbalized Machine Learning: Revisiting Machine Learning with Language Models

Anonymous Authors

Abstract
Motivated by the large progress made by large language models (LLMs), we introduce the framework of verbalized machine learning (VML). In contrast to conventional machine learning 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 machine learning 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, (2) automatic model selection and (3) interpretable learner update.

1. Introduction
The unprecedented success of large language models (LLMs) has changed the way people solve new problems in machine learning. Compared to 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 contains in-context examples and well-performing instructions. The two ways of problem-solving lead to an intriguing comparison. Traditionally, we optimized a neural network in a continuous parameter space using gradient descent, while in the new approach, we are optimizing the input prompt of an LLM in a discrete natural language space. Can a pretrained LLM act as a function that is parameterized by its natural language 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 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. Specifically, we propose to view the input text prompt of LLMs as the model parameters that are being learned. However, it also introduces additional difficulties to the optimization of such a natural language parameter space. Inspired by previous work [1, 11] 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 such that the training objective can be reached.

Compared to conventional numerical machine learning, VML 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.

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 the 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 contribution includes: (1) We formulate the framework of verbalized machine learning, where pretrained language models are viewed as function approximators parameterized by its text.
prompt. Then we revisit some classical machine learning problems and show that VML is able to solve them. (2) 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. (3) We conduct an empirical study 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.

2. Related Work

Language models are used to perform planning for embodied agents [18, 26, 12, 14], such that they can follow natural language instruction to complete complex tasks. More recently, LLMs are used to solve optimization problems [27]. Specifically, the LLM generates new solution to the optimization problem from the prompt that contains previously generated solutions with their loss values. The LLM optimizer in [27] shares a high-level similarity to our work, as we both want to solve an optimization problem with LLMs. The key difference to [27] is our function approximation view of LLMs, which enables us to revisit classical machine learning problems and solve them through natural language in the VML framework. There are many prompting methods [23, 31, 32, 22, 28, 29, 25] designed to elicit the reasoning ability of LLMs. To reduce the hand-crafting efforts in designing good prompts, automatic prompt optimization [31, 32, 27, 16, 24, 3, 13, 15, 19] has been proposed. Unlike prompt optimization where the text prompt is optimized without changing its semantic meaning, VML updates its language-based model parameters by adding or modifying the model prior information, making the learner model fully interpretable about its prediction.

3. Verbalized Machine Learning

3.1. Natural Language as the Model Parameter Space

Figure 1 gives a comparison between the classic numerical machine learning and VML. In the VML framework, both data and model are represented by a unified token-based format, while numerical machine learning treats data and model parameters differently. VML parameterize a machine learning model with natural language. More formally, VML places a strong constraint to the model parameters \( \theta = \{ \theta_1, \theta_2, \cdots, \theta_t \} \in \Theta_{\text{language}} \) for the interpretability, where \( \theta \) is a text token sequence, \( \theta_t \in \mathcal{A}, \forall t \) is some text token from a large token set \( \mathcal{A} \), and \( \Theta_{\text{language}} \) denotes the set of all natural language sequences that human 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 therefore we generally have, \( \prod \mathbb{P}(\theta_t|\theta_{t-1}) \neq \prod \mathbb{P}(\theta_t) \), while numerical machine learning typically has a i.i.d. model parameter space: \( \prod \mathbb{P}(\theta_t|\theta_{t-1}) = \prod \mathbb{P}(\theta_t) \); and (3) human-interpretable: the natural language that characterizes the model should be human-interpretable.

One significant advantage 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 is modified. Our empirical evidences also support our interpretability claim, as we find that the model parameters \( \theta \) are typically a language description of the underlying pattern that the model discovers from the training data.

3.2. Language Models as Function Approximators

The core idea behind VML is using a language model to act as a function approximator parameterized by its natural language prompt. Specifically, we denote the language model as \( f(x; \theta) \) where \( x \) is the input data and \( \theta \) is the function parameter. Both \( x \) and \( \theta \) are represented with text tokens. In VML, \( f(\cdot) \) is actually a frozen language model that is pretrained on a large corpus of text (e.g., Llama-3 [20], ChatGPT). If we consider a static function, then we can set the temperature parameter of the LLM as zero, which will theoretically make the output deterministic. If we set the temperature high, then \( f(x; \theta) \) can be viewed as sampling a value from some distribution.

We revisit how a classical machine learning problem is formulated in the VML framework. Suppose we have \( N \) data points \( \{ x_n, y_n \}_{n=1}^{N} \) in total, where \( x_n \) is the data vector and \( y_n \) is the target value. As an example, we consider a least square regression problem using the LLM-parameterized function, i.e., for \( \theta \in \Theta_{\text{language}} \):

\[
\min_{\theta} \ell_{\text{regress.}} := \frac{1}{2N} \sum_{n=1}^{N} (y_n - f_{\text{model}}(x_n; \theta))^2,
\]

where minimizing the objective function with respect to the discrete token-based model parameters \( \theta \) is actually quite difficult. Back-propagating gradients through discrete variables (e.g., policy gradients, gumbel-softmax [6]) is typically known to be inefficient and sub-optimal.

3.3. Training by Prompt Optimization

Because the model parameters \( \theta \) in VML is essentially a text prompt, optimizing \( \theta \) is effectively a prompt optimization problem. Different from the prompt optimization problem [32] where the goal is to produce a generic prompt
Verbalized Machine Learning: Revisiting Machine Learning with Language Models

Figure 2: An overview of iterative optimization and text prompt templates of the learner and the optimizer in the regression example.

Algorithm 1 Training in VML

```
Initialize model parameters \( \theta \), iteration number \( T' \), batch size \( M \) and optimizer parameters \( \psi \);
for \( i = 1, \ldots, T' \) do
   Sample \( M \) training data \( x_1, \ldots, x_M \);
   for \( m = 1, 2, \ldots, M \) do
      \( \hat{y} = f_{\text{model}}(x_m; \theta) \);
   end
   \( \theta = f_{\text{opt}}(\{x_i, \hat{y}, y_i\}; \theta; \psi) \);
end
```

where \( \eta \) is the learning rate, and the constraint is to ensure that the updated model parameters are still in the human-interpretable 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 \((x, y)\) and the current model parameters \( \theta \). Then we directly approximate the next-step model parameters using another pretrained language model denoted by \( f_{\text{opt}}(x, \hat{y}, y, \theta; \psi) \) where \( \hat{y} \) is the model prediction from the learner \( f_{\text{model}} \). \( \psi \) denotes the optimizer parameters that characterize the optimization settings, and we can use language to specify the update speed, the momentum, 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 quite effective in [27]. More importantly, the performance of our VML framework gets better as the instruction-following ability of LLMs gets stronger. An overview of the text prompt templates in VML is given in Figure 2. The algorithmic procedure is given in Algorithm 1.

Using an LLM as the optimizer offers several 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 4.1 as an example). Second, the optimizer can provide detailed explanation of why a particular update should be performed, which helps us to understand the inner working mechanism of the optimization process. Third, the LLM-paramterized optimizer allows users to interact with it directly. This not only helps us to trace model failures, but it also allows us to inject prior knowledge to improve optimization.

4. Application and Case Study

We demonstrate the features and advantages of VML by revisiting some simple yet classical machine learning tasks including regressions and classifications. Tasks other than polynomial regression are included in Appendix B. In these tasks, we are given \( \mathcal{D}_{\text{train}} = \{x_n, y_n\}_{n=1}^N \), and we want to find \( \theta^* \) such that \( f_{\text{model}}(x; \theta^*) \) best describes the mapping \( x \rightarrow y \). Our experiments below show in detail how VML is...
able to solve these tasks and find \( \theta^* \).

**Experiment setups.** We use the instruction-tuned Llama-3 70B [20] 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 step of optimization (see Figure 2 (right) as an example), which corresponds to 10 steps per epoch of training. To evaluate performance, we look at the training loss, and the model predictions in both the interpolation and extrapolation settings. As for classifications, we use additional test sets consist of 20 data points, and evaluate the training and testing accuracies. During optimization, inspired by the idea of momentum from classical machine learning optimization, we also provide the last step (i.e., one step only) of the optimization history to stabilize training.

**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., \( \theta_{i - 1} \)), and the optimizer output for the updated \( \theta_i \). For \( i > 1 \), the full model parameter before optimization is \( \theta_{i - 1} = \{ \theta_0, \theta_{i - 1} \} \), but in our figures below we only show the \( \theta_{i - 1} \) to save space.

### 4.1. 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, \( \theta_0 \) is initialized 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 the training is effective and does converge. The subplot (b) and (c) show the detail of the model and optimization at step 1, 2 and 3. At step 1, model\(_0\) randomly guesses the outputs. The optimizer\(_1\) says that it notices \( y \) has a large range than \( x \), and they seems to have positive correlation, therefore, it updates model\(_1\) to be a simple linear regression 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 realizes the linear model assumption oversimplifying the relationship between the \( x \) and \( y \), it notices the a non-linear relationship between \( x \) and \( y \), and to capture this curved relationship it will use a quadratic model. This results in a much better model and leads to a large decrease in the training loss. At step 3, optimizer\(_3\) switches from model class selection to fitting the identified quadratic model. The resulting model\(_3\) closely fits the ground truth.
Verbalized Machine Learning: Revisiting Machine Learning with Language Models

References


Verbalized Machine Learning: Revisiting Machine Learning with Language Models


[27] 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


Appendix

Table of Contents

A Discussions, Insights, and Limitations .................................................. 8

B More Case Studies
   B.1 Linear Regression .............................................................................. 9
   B.2 Sinusoidal Regression ....................................................................... 10
   B.3 Two Blobs Classification .................................................................. 11
   B.4 Two Circles Classification ................................................................ 12
   B.5 Comparison Between Prompt Optimization and VML ....................... 13

C Effect of Accurate Loss Feedback ............................................................ 14

D Numerical Error in LLMs ......................................................................... 15

E Connection between Prediction Variance and Model Parameters in VML
   E.1 From Vague to Concrete Model Parameters ...................................... 17
   E.2 Semantic Invariance of Model Parameters ....................................... 18

F Complete Training Template at Initialization
   F.1 Linear Regression.................................................................................. 20
   F.2 Polynomial Regression ......................................................................... 21
   F.3 Sinusoidal Regression ......................................................................... 22
   F.4 Two Blobs Classification ..................................................................... 23
   F.5 Two Circles Classification .................................................................. 24
   F.6 Text classification ................................................................................ 25
A. Discussions, Insights, and Limitations

VML as a unified 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 [8] for random variables, recurrent network [5] for sequences, graph network [7] for graphs, and convolution network [10] 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 $\theta_{\text{prior}}$ (i.e., some constant prefixed text that describes the inductive bias) with the model parameters $\theta$. Therefore, the final model parameters will be $\{\theta_{\text{prior}}, \theta\}$ where only $\theta$ is learnable and $\theta_{\text{prior}}$ is provided by users.

Difference between VML and prompt optimization. Both VML and prompt optimization aims to automatically produce a text prompt towards some target, but VML differs from existing prompt optimization works (e.g., [32, 16]) in a substantial way. First, VML aims to automatically discover a verbalized data pattern that act as the model parameters for the LLM learner, while prompt optimization seeks a generic instruction without changing the original meaning to elicit the best downstream question-answering performance. We will qualitatively compare the difference of their learned prompts in the experiment section. Second, prompt optimization can be viewed as a building block for VML, as its techniques can be naturally adapted for the training of VML.

The effectiveness of in-context learning suggest the feasibility of VML. In-context learning [4] is a training-free framework for instructing LLMs to accomplish some tasks. VML shares a high-level similarity with in-context learning in the sense that they both aim to add more information to the text prompt. In-context learning typically provides more demonstrations and examples for the downstream tasks, while VML replies on the optimizer LLM to discover and summarize patterns from the data. [21] shows that in-context learning can be roughly viewed as performing the gradient descent on the in-context examples. This result suggests that LLM with text prompts can be a powerful function approximator, validating that VML can learn a good model by updating its model parameters.

VML enables interpretable knowledge discovery. Because the model parameters $\theta$ is already natural language, it is easy to understand the underlying pattern that leads to the prediction and the decision rules that the model use. Unlike numerical machine learning, this property makes VML possible to discover novel knowledge that human can also learn from.

VML is “the von Neumann architecture” in machine learning? Conventional numerical machine learning usually treats the model parameters and the data differently, which is 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. This distinction makes VML an interesting direction to explore.

Concluding Remarks and Limitations. Our paper introduces a verbalized way to perform machine learning and performs a few simple case studies on regression and classification tasks. The experiments show that VML can effectively perform these classical machine learning, validating the potential of language models as function approximators. Despite the empirical effectiveness, there are a few limitations that remain to be addressed. First, the training in VML still suffers from a relatively large variance. This is partially due to the stochasticity from the LLM inference. Second, the output numerical error in LLMs result in inevitable fitting error. Concretely, even if the LLM understands correctly the underlying symbolic expression, there is still an output numerical error when performing the inference on specific input values. This also suggests the intrinsic difficulty within LLMs to properly understand numbers (see [17, 30]). Third, the input data dimensionality and batch size are limited by the context window of LLMs, preventing VML from processing high-dimensional data or optimizing with a large batch size.
B. More Case Studies

**Compute.** The LLM is run on a node of $8 \times A100$ using the inference engine provided by vLLM [9]. During each step $(i)$ of training, we query the LLM 10 times for evaluating the model $f_{\text{model}}(x; \theta_{i-1})$ over a batch, and 1 time for requesting the newly optimized $\theta_i$. We also evaluate the entire test set at each step, which depends on the size of the evaluation set, we might query the LLM between 20 to 100 times. Overall, for the regression tasks, they take around 10 minutes for each epoch of training. For the classification tasks, they take around 16 minutes for each epoch of training. The additional 6 minutes of overhead is due to evaluating the grid for the background of the decision boundary.

### B.1. Linear Regression

To optimize the pattern descriptions, I will introduce a scaling factor to 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.

Reasoning:

The current model outputs are not close to the target outputs, indicating that the model 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, suggesting that the scaling factor is too small, hence it will increase the it. Similarly, at step 15, I will add this value to the current bias term of 1.05 to get a new bias term of 3.28.

I will also analyze the relationship between the inputs and target outputs to determine the scaling factor. Upon examining the data, I notice that the inputs are generally smaller than the target outputs, indicating that the current pattern descriptions are not effective. Upon examining the data, I notice that the inputs and target outputs are generally 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: $y = 4.2x + 1.05$

We generate $D_{\text{train}}$ 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 $\theta_0$ 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 the training does improve the model, and it does converge. The subplot (b) and (c) show the detail of the model and optimization at step 1, 3 and 15. At step 1, since $\theta_0$ only contain the definition of 1-D regression task, the model $\theta_0$ is randomly guessing (see the dashed line). The optimizer $\theta_1$ says that it notices a linear relationship between the input and target outputs, hence introduce the a linear regression model to capture such relationship, which results in model 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 $\theta_3$ says it notices the model outputs are generally smaller than the target, suggesting the scaling factor is too small, hence it will increase the it. Similarly, at step 15, optimizer $\theta_{15}$ also says it notices the model overestimates the target, hence, it reduces the scaling factor. We can see from (b) that the resulting model closely approximates the ground truth.

![Figure 4](image-url)
B.2. Sinusoidal Regression

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:

\[
\text{Output} = 1.05 \times \sin(0.98 \times \text{input}) + 2.0.
\]

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.

We generate \( D_{\text{train}} \) 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 \( \theta_0 \) 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 \( \theta \) 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)). But adding the same prior to Llama-3 is not as effective (see (b,c; mid)), indicating the capability of VML depends on the capability of the underlying LLM.
B.3. Two Blobs Classification

We generate a linearly separable $D_{train}$ from two blobs on a 2-D plane. $\theta_i$ is initialized by *only* specifying that the task is binary classification on a 2-D plane (see Figure 6(c) Step 1). The subplot (a) shows that the training is effective and able to converge. At step 1, optimizer 1 *says* its inspection of current batch of data has the pattern that points with $x > 0$ belong to class 2, and points with $x < 0$ belong to class 1, hence it updates model 1 to have a linear decision boundary at $x = 1$, 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 what does optimizer 2 *say*. From (c) Step 2, we see that optimizer 2 *says* model 1 is already quite simple and accurate, but it wants to further improve the model and utilize the new information from the current batch. Guided by this reasoning, model 50 becomes a very deep decision tree, and the decision boundary has a reasonable margin towards the data (see Figure 6(b, c, right)).
B.4. Two Circles Classification

![Training loss dynamics](image1)

![Visualization of the learned decision boundary at Step 1, Step 40 and Step 41](image2)

![Model parameters and optimizer output at Step 1, Step 40 and Step 41](image3)

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 \((x)^2 + (y)^2 \leq 0.65\), then output 1, else output 0.

---

Figure 7: Non-linearly separable two circles classification with a prior in \(\theta\). (a; dashed) and (c; bottom right) also show results without the prior.

We generate a non-linearly separable \(D_{train}\) by creating two circles, one within the other, for the two classes. Besides the definition of binary classification on a 2-D plane, we also add a sentence to encode our inductive bias that the decision boundary is a circle into \(\theta_0\) (see Figure 7(c) Step 1). At step 1, \(\text{optimizer}_1\) utilizes the prior information, and updates \(\text{model}_1\) to have a circle decision boundary. For the rest of the training step, the optimizer mainly tries to find a good fit for the radius and the center of the decision boundary. At step 41, \(\text{optimizer}_{41}\) says \(\text{model}_{40}\) seems to be a good fit for the data, and no changes are needed, hence, it uses the same \(\theta_{40}\) for \(\text{model}_{41}\). Without the prior, VML can also learn a good model, but the performance show large variance at the beginning of the training (see Figure 7(a; dashed)) due to the model class selection process similar to Figure 4(a). Figure 7(c; bottom right) shows the resulting \(\theta_{40}\) without the prior, which is a decision tree.
**B.5. Comparison Between Prompt Optimization and VML**

**Automatic Prompt Engineer**

calculate a mathematical function. Can you guess what the function is?

*Note:* The function is not a polynomial function.

*Hint:* The function is a well-known mathematical function.

*Edit:* The inputs and outputs are accurate up to 2 decimal places.

**Automatic Prompt Engineer**
give a 1 if the name was feminine and a 0 if it was masculine. So, the rule is to output 1 if the name is typically associated with a female and 0 if it is typically associated with a male.

This got

**Model parameters in VML**

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.

(a) Linear regression

(b) Text classification

Figure 8: VML versus a prompt optimization method (Automatic Prompt Engineer [32]).

To differentiate VML from prompt optimization, we qualitatively compare VML to a popular prompt optimization method called Automatic Prompt Engineer (APE) [32] on two tasks.

**Linear regression** as in Appendix B.1. Figure 8(a) shows that the result from APE is vague and general. Such descriptions can be easily derived by human through visual inspection of the data, and it does not learn deeper insight from the data, whereas VML is able to learn useful new information that is difficult to derive by visual inspection through the data.

**Text classification.** Adopted from the Google BIG-bench[2], the task is to classify whether a name is more likely to be associated to female or male. Figure 8(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.
C. Effect of Accurate Loss Feedback

The VML algorithm at Algorithm 1 specifies that the arguments for $f_{\text{opt}}(\cdot)$ consist of the inputs $x$, the predictions $\hat{y}$, the targets $y$, the current model parameter $\theta_{i-1}$ and the optimizer configurations $\psi$. 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 9 shows that having such accurate loss feedback might not help, and might even decrease the performance. One explanation is that the single loss value itself does not contain too much information. 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.
D. Numerical Error in LLMs

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 10 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 11 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 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.
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 $2\pi$ (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 $\pi$ (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 $2\pi$ (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.

Figure 12: Function evaluations based on the natural language description of the corresponding symbolic sine function.

Figure 12 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 underperforms 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.
E. Connection between Prediction Variance and Model Parameters in VML

E.1. From Vague to Concrete Model Parameters

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 13 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 \( \approx 1 \)) 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.
E.2. Semantic Invariance of Model Parameters

In natural language, there are different ways to describe one concept. 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 14 shows that results on Llama-3 70B for six different but semantically invariant descriptions of the model from Figure 4(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 $\theta_{15}$ from Figure 4(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 14(a,b,c) are similar, and have small variance across the 10 runs. The results in Figure 14(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.
F. Complete Training Template at Initialization

F.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: **

[SData]

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): **

[[SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData]]

** 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: **

[[SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction]]

** The target outputs: **

[[SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth]]

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:

```
Explanations: [Your step-by-step analyses and results]
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 15: Prompt templates of VML for the learner and optimizer for the linear regression (LLama-3-70B without prior).
F.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: **
[SData]

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): **
[[SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData]]

** 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: **
[[SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction]]

** The target outputs: **
[[SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth]]

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 16: Prompt templates of VML for the learner and optimizer for the polynomial regression (Llama-3-70B without prior).
F.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]

** The target outputs: **
[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 17: Prompt templates of VML for the learner and optimizer for the sinusoidal regression (GPT-4o with prior).
F.4. Two Blobs Classification

** 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., \([\text{class 1 prob.} \ \text{class 2 prob.}]\). The sum of the vector MUST be 1.0.

** Input: **

\([\text{Data}] \ [\text{Data}] \ [\text{Data}] \ [\text{Data}] \ [\text{Data}] \ [\text{Data}] \ [\text{Data}] \ [\text{Data}] \ [\text{Data}]\)

** 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., \([\text{class 1 prob.} \ \text{class 2 prob.}]\). 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 (\([\text{class 1 prob.} \ \text{class 2 prob.}]\)): **

\([\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}] \ [\text{Prediction}]\)

** The targets (\([\text{class 1 prob.} \ \text{class 2 prob.}]\)): **

\([\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}] \ [\text{GroundTruth}]\)

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!!!****]

** Output: **

\([\text{New Model Descriptions}]\)

Please ONLY reply according to this format, don't give me any other words.

Figure 18: Prompt templates of VML for the learner and optimizer for the two blobs classification (Llama-3-70B without prior).
F.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: **
[SData]

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: \([x, y]\)):**
[[SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData]]

** 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:**
[SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction]

** The targets:**
[SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth]

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 19: Prompt templates of VML for the learner and optimizer for the two circles classification (Llama-3-70B with prior).
F.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: **

[SData]

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): **

[[SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData] [SData]]

** 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: **

[[SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction] [SPrediction]]

** The targets: **

[[SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth] [SGroundTruth]]

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 20: Prompt templates of VML for the learner and optimizer for the text classification (Llama-3-70B without prior).