`DLPO` : Towards a Robust, Efficient, and Generalizable Prompt Optimization Framework from a Deep-Learning Perspective

Dengyun Peng♠* **Yuhang Zhou**♠◇* **Qiguang Chen**♠* **Jinhao Liu**♠
Jingjing Chen◇ **Libo Qin**♣† **Wanxiang Che**♠†

♠ Research Center for Social Computing and Interactive Robotics
♠ Harbin Institute of Technology, China
♣ School of Computer Science and Engineering, Central South University, China
◇ Fudan University, China

{dypeng,qgchen}@ir.hit.edu.cn, ralph.yh.zhou@gmail.com
lbqin@csu.edu.cn, car@ir.hit.edu.cn

Abstract

Large Language Models (LLMs) have achieved remarkable success across diverse tasks, largely driven by well-designed prompts. However, crafting and selecting such prompts often requires considerable human effort, significantly limiting its scalability. To mitigate this, recent studies have explored automated prompt optimization as a promising solution. Despite these efforts, existing methods still face critical challenges in ***robustness, efficiency, and generalization***. To systematically address these challenges, we first conduct an empirical analysis to identify the limitations of current reflection-based prompt optimization paradigm. Building on these insights, we propose 7 innovative approaches inspired by traditional deep learning paradigms for prompt optimization (`DLPO`), seamlessly integrating these concepts into text-based gradient optimization. Through these advancements, we progressively tackle the aforementioned challenges and validate our methods through extensive experimentation. We hope our study not only provides valuable guidance for future research but also offers a comprehensive understanding of the challenges and potential solutions in prompt optimization. Our code is available at https://github.com/sfasfaffa/DLPO.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable achievements across diverse applications (Brown et al., 2020; Ouyang et al., 2022; Zhao et al., 2023; Qin et al., 2024a; Zhuang et al., 2023). Notably, the performance of LLMs highly relies on well-crafted prompts (Zhu et al., 2023; Li et al., 2024). Given their sensitivity to prompts, selecting optimal prompts is crucial for maximizing their performance in downstream tasks, a process known as Prompt Engineering (Reynolds
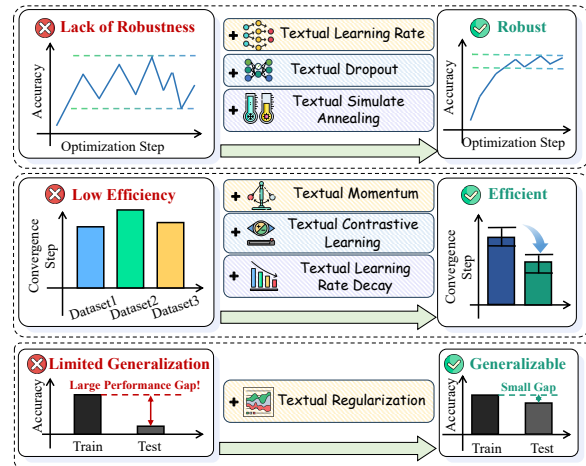

Figure 1: Comparison between traditional reflection-based prompt optimization methods and `DLPO`, which incorporates **7 innovative approaches** to progressively enhance the robustness, efficiency, and generalizability of prompt optimization.

and McDonell, 2021; Sahoo et al., 2024), which requires substantial human effort (Zamfirescu-Pereira et al., 2023). To address this, researchers have developed automated prompt optimization (PO) techniques, such as reinforcement learning-based methods (Deng et al., 2022; Zhang et al., 2022; Diao et al., 2022), search-based methods (Prasad et al., 2022; Pryzant et al., 2023; Zhou et al., 2022), all aiming to improve efficiency and stability. Among these methods, a dominant paradigm leverages LLMs' reflection capabilities, optimizing prompts based on feedback from external environments (Wang et al., 2023; Tang et al., 2024). Following Pryzant et al. (2023) and Yuksekgonul et al. (2024), this paradigm draws inspiration from neural network training, reinterpreting key concepts: treating prompts as model parameters, output metrics as loss, and LLM reflection as gradients within textual expression. Such paradigms has proven to be a powerful and widely adopted tool for various downstream tasks (Zhou et al., 2024; Mañas et al.,

---

* Equal Contribution
† Corresponding Author

2024; Du et al., 2024).

Despite significant advancements in automated PO, our preliminary exploration reveals that existing methods encounter practical limitations that hinder their widespread adoption. These challenges can be categorized into three key issues, as illustrated in Fig.1: (1) *Lack of Robustness.* Current update methods often exhibit substantial oscillations and instability during the update process. (2) *Low Efficiency.* Existing PO strategies require massive iterations to achieve ideal prompts. (3) *Limited Generalizability.* While existing methods perform well on in-domain tasks present in training data, they struggle with distributionally out-of-domain tasks, which extremely restricts their applicability in real-world scenarios.

To address this, we first perform a systematic review of the prevailing reflection-based PO paradigm to identify its limitations. Inspired by traditional deep learning approaches, we propose novel techniques for DLPO to enhance its PO capabilities. Traditional machine learning (ML) paradigm considers prompts as a single module, and relies on monotonic and uncontrollable gradient optimization process. In contrast, we view the semantic dependencies within complex prompts as a neural network, enabling the application of advanced deep learning methods. (1) Unlike traditional ML approaches, our method employs Textual Dropout, which randomly discards certain prompt sentences to improve robustness. Additionally, we use Textual Learning Rate and Textual Simulated Annealing to better control the optimization direction, enhancing robustness over monotonic gradient optimization. (2) In deep learning, optimization speed often depends on the gradient optimization difficulty. To accelerate convergence, we apply Textual Learning Rate Decay, Textual Momentum, and Textual Contrastive Learning for more efficient gradient optimization. (3) Unlike the single prompt modularization in the ML paradigm, our method integrates Textual Regularization to prevent over-complication of prompt sentences, ensuring that the model remains concise. The correspondence between all seven methods we propose and classical deep learning methods is shown in Table 1.

To evaluate the effectiveness of our proposed approach, we conduct extensive experiments on five datasets covering multiple tasks. The results indicate that our method significantly enhances the stability, efficiency, and generalizability of prompt optimization. Notably, it outperforms the previous
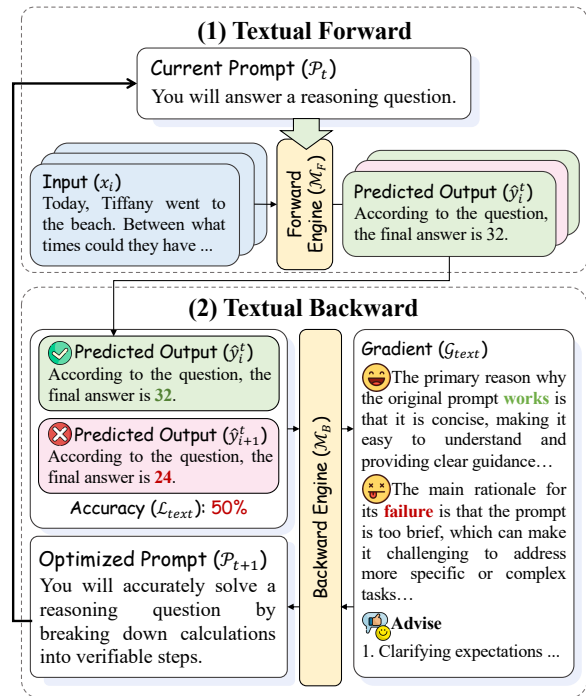

Figure 2: Current reflection-based paradigm for prompt optimization.

state-of-the-art method by 8.1% and even surpasses manually designed prompts, demonstrating its superiority in automated prompt optimization.

The contributions of this work are summarized as follows:

- We first identify key challenges in robustness, efficiency, and generalization in LLM-based prompt optimization, revealing the limitations of existing methods through both theoretical and empirical analysis.

- Drawing inspiration from traditional deep learning techniques, we propose DLPO to enhance robustness, efficiency, and generalization by incorporating 7 novel text-based gradient optimization strategies.

- We conduct extensive experiments across diverse datasets to validate our approach, providing a comprehensive understanding of prompt optimization challenges and best practices.

2 Problem Formalization

2.1 Prompt Optimization

As shown in Figure 2, the paradigm of prompt optimization leverages LLMs as optimizers by refining and enhancing prompts based on feedback from external environments. Formally, we consider a

Traditional Methods	DLPO Methods	Description
Learning Rate	Textual Learning Rate (TLR)	TLR controls the number of sentence modifications per step to stabilize updates.
Dropout	Textual Dropout (TDO)	TDO randomly skips sentence updates to reduce overfitting and preserve beneficial modifications.
Simulated Annealing	Textual Simulated Annealing (TSA)	TSA uses training accuracy as energy, accepting suboptimal solutions probabilistically.
Learning Rate Decay	Textual Learning Rate Decay (TLRD)	TLRD enables early-stage exploration followed by gradual refinement of prompts.
Momentum	Textual Momentum (TMnt)	TMnt utilizes past gradients to smooth updates and enhance efficiency.
Contrastive Learning	Textual Contrastive Learning (TCL)	TCL differentiates high- and low-quality prompts to encourage effective patterns.
L1/L2 Regularization	Textual Regularization (TRegu)	TRegu removes redundant phrases and simplifies sentence structures for better generalization.

Table 1: A brief comparison of traditional deep learning methods and their textual counterparts in DLPO. Some methods adapt numerical optimization concepts to the text space through sentence-level operationalization.

training dataset $\mathcal{D} = (x_i, y_i)_{i=1}^n$ drawn from a joint distribution $\mathbb{D}(X, Y)$.

Textual Forward Given an input-output pair (x_i, y_i) and current prompt $\mathcal{P}_t$, the LLM $\mathcal{M}_F$, acts as forward engine, generating a predicted output:

$$\hat{y}_i^t = \mathcal{M}_F(y_i | x_i, \mathcal{P}_t). \quad (1)$$

Textual Backward To quantify the discrepancy between the predicted output $\hat{y}_i^t$ and the ground truth y_i, we define the textual loss function as:

$$\mathcal{L}_{text} = \mathcal{F}_{loss}(\{\hat{y}_1^t, y_1\}, \ldots, \{\hat{y}_i^t, y_i\}, \ldots), \quad (2)$$

where $\mathcal{F}_{loss}(\cdot)$ denotes the human-defined loss function, such as rule-calculated accuracy or model-generated score.

To guide the improvement of the prompt $\mathcal{P}_t$, textual gradient is defined to determine the direction of optimization. Here, the LLM acts as an implicit gradient estimator, computing the gradient through model reflection:

$$\mathcal{G}_{text} = \frac{\partial \mathcal{L}_{text}}{\partial \mathcal{P}_t} = \mathcal{M}_B(\mathcal{G} | x_i, y_i, \mathcal{P}_t, \mathcal{L}_{text}), \quad (3)$$

where $\mathcal{M}_B$ denotes the LLM that functions as a backward engine. It is always considered more powerful compared to the forward engine.

After textual gradient computation, the prompt $\mathcal{P}_t$ is iteratively updated using the textual gradient to minimize the loss. At this stage, the LLM functions as the optimizer, generating an improved prompt $\hat{\mathcal{P}}_{t+1}$ as:

$$\hat{\mathcal{P}}_{t+1} = \mathcal{M}_B(\mathcal{P}_{t+1} | x_i, y_i, \mathcal{P}_t, \mathcal{G}_{text}). \quad (4)$$

2.2 Prompt Application

Once optimized, the prompt is applied to previously unseen test data x' to generate the predicted output:

$$\hat{y}' = \mathcal{M}(y' | x', \mathcal{P}^*). \quad (5)$$

where $\mathcal{P}^*$ refers to the optimized prompt. Our primary framework for implementing prompt optimization is TextGrad (Yuksekgonul et al., 2024), a widely recognized prompt optimization framework. It modularly implements the entire pipeline outlined in this section, making it particularly well-suited for our analysis.

3 Experiment Setups

We utilize several benchmarks to evaluate the effectiveness of our proposed methods, including GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), BigGSM (BGSM) (Chen et al., 2024), BigBenchHard Object Counting (BBH) (Suzgun et al., 2022), and MGSM (Shi et al., 2022). The detailed descriptions can be found in Appendix C.1.

4 Preliminary Analysis

In this section, we explore the following three key drawbacks for current reflection based prompt optimization paradigm.

4.1 Lack of Robustness

The robustness of an optimization process is a critical prerequisite for both convergence and generalization. In this context, we define robustness as the stability of the optimization process. As illustrated in Fig. 3 (a,b), the results reveal significant instability. Specifically, variations in random seeds lead to substantial discrepancies in convergence, with the variance at the final step reaching 20.8% for the GSM8K environment. Moreover, in the BBH, the current paradigm fails to achieve convergence even by the end of the training process.

4.2 Low Efficiency

Efficiency is defined here as the convergence speed during prompt optimization. As illustrated in Fig. 3

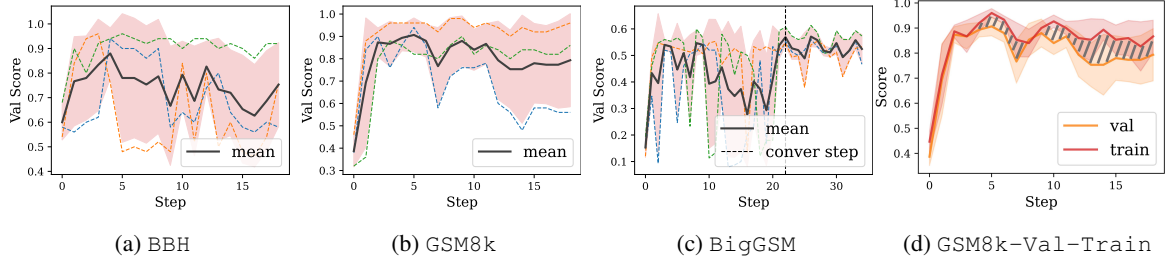(a) BBH   (b) GSM8k   (c) BigGSM   (d) GSM8k-Val-Train

Figure 3: **a**, **b**, **c** respectively show the validation-set accuracy of 3 different seeds and their mean values and standard deviations on BBH, GSM8K, and BigGSM environments. The mean values are represented by black solid lines, and the standard deviations are indicated by red shaded areas. **d** shows the **Training-set** and **Validation-set** mean accuracy results of 3 different seeds on GSM8K, along with their standard deviations. To make the image clearer, we use $\frac{1}{2}$ standard deviation as the shaded area.

(c), we aim for the prompt optimizer to achieve rapid convergence to a stable value. In contrast, the current approach necessitates over 20 iterations to attain its peak performance and reach stabilization. This stark difference underscores the significant inefficiency of the existing method in comparison.

4.3 Limited Generalizability

We define *generalizability* as the degree to which LLMs can generalize to test sets or diverse data distributions within the same task domain. As shown in Fig. 3 (d), the grey shadow reveals a large discrepancy between the validation and training set scores (approximately 10% accuracy at the optimal point), suggesting that the current update method lacks sufficient generalization capability.

5 Exploration

In this section, we solve the above three problems using methods inspired by traditional Deep Learning step by step. The detailed implementation of our methods can be found in appendix B.

5.1 Exploration for Robustness

The robustness of optimization is one of the most critical issues in subsequent work. Only by ensuring the robustness of optimization can other problems and methods be deemed worthy of consideration and meaningful.

We meticulously analyze the optimized sequence of prompts and observe that the variations between updates are highly significant. After just a single update, the prompts appear entirely different compared to their previous state. To address this issue, we propose three effective methods: Textual Learning Rate (TLR) , Textual Dropout (TDO) and Textual Simulated Annealing (TSA).
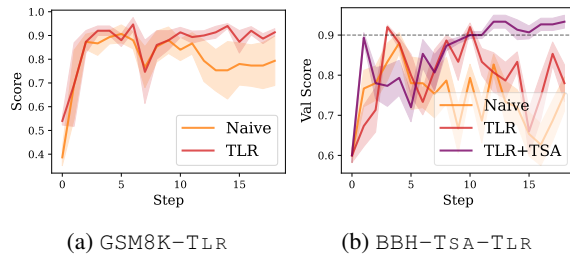

(a) GSM8K-TLR   (b) BBH-TSA-TLR

Figure 4: **a**, mean results of 3 different seeds for TLR+TDO and Naive on GSM8K. We use $\frac{1}{2}$ of the standard deviation as the shaded area. **b**, mean results of 3 different seeds for TSA+TLR+TDO, TLR+TDO and Naive on BBH. To make the image clearer, we use $\frac{1}{4}$ of the standard deviation as the shaded area.

5.1.1 Textual Learning Rate & Dropout

Textual Learning Rate (TLR)   Traditional ML-based PO often disregards the gradient's magnitude, impeding convergence to the optimal solution. To address this, drawing inspiration from DL, we propose a Textual Learning Rate to regulate the extent of gradient updates. Specifically, each modification, deletion, or addition of a sentence in a prompt constitutes an update unit. The TLR value, denoted as $\mathcal{R}$, sets an upper limit on the number of update units per step. If an LLM attempts to exceed $\mathcal{R}$, we prompt LLMs to prioritize impactful changes while discarding non-essential ones.

Textual Dropout (TDO)   Traditional ML-based PO treats the prompt as an indivisible whole, making local optimization challenging and leading to large updates that compromise robustness. To address this, we adopt a DL perspective, viewing the prompt as a structured semantic network. Specifically, we introduce Textual Dropout (TDO), which selectively drops or skips sentence updates to pre-
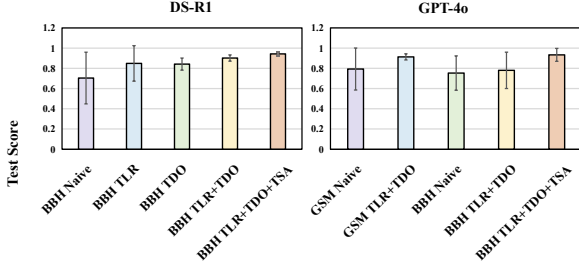
Figure 5: Comprehensive evaluation of TLR, TDO, and TSA methods on GSM8K and BBH object counting benchmarks with different optimizers (GPT-4o and DeepSeek-R1).

vent full parameter optimization.

Formally, given a prompt $\mathcal{P}_t$ with $\mathcal{S}$ sentences and a dropout rate p, the number of preserved sentences $\mathcal{K}$ is:

$$\mathcal{K} = \lceil p \cdot \mathcal{S} \rceil. \qquad (6)$$

The updated prompt $\hat{\mathcal{P}}_{t+1}$ is:

$$\hat{\mathcal{P}}_{t+1} = \mathcal{P}_{\text{preserved}} \oplus \mathcal{P}_{\text{updated}},$$

where $\oplus$ denotes merge operation. Here, $\mathcal{P}_{\text{preserved}}$ consists of the $\mathcal{K}$ preserved sentences, while $\mathcal{P}_{\text{updated}}$ includes the modified sentences based on the textual gradient $\mathcal{G}_{\text{text}}$.

Since both TLR and TDO regulate gradient updates at the sentence level and encourage a certain degree of restriction on the magnitude of single-step updates, we apply them jointly to evaluate their effectiveness. As shown in Fig. 4 (a,b), experiments on GSM8K and BBH demonstrate that TLR+TDO outperforms the naive approach and significantly enhances optimization robustness. Fig. 5 further validate the independent effectiveness of each method across different optimizer models (DeepSeek-R1 and GPT-4o). More details are presented in the App. A.

5.1.2 Textual Simulated Annealing TSA

To further enhance the gradient optimization in DL views, we propose Textual Simulated Annealing scheme to guarantee stable update during training by dynamically adjusting prompts based on their accuracy to effectively control the optimization direction. Simultaneously, to prevent the model from getting stuck in local optima, TSA encourages the model to accept suboptimal solutions with a certain probability. Specifically, we define training set accuracy as the "Energy" function ($\mathcal{E}(\cdot)$), and compare the accuracy before and after each update. If accuracy decreases, the suboptimal solution is accepted with a probability determined by:
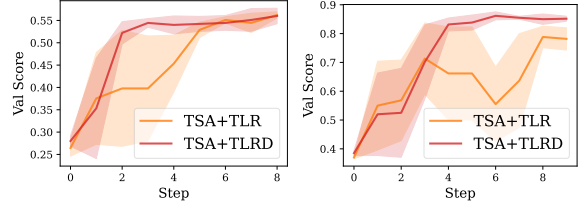
(a) BigGSM-TLRD   (b) MGSM-TLRD

Figure 6: **a**, **b**, mean results of 3 different seeds for TSA+TLRD and TSA+TLR on validation set of BigGSM and MGSM environment. We use $\frac{1}{2}$ of the standard deviation as the shaded area.

$$P(\Delta\mathcal{E}, \mathcal{T}) = \exp\left(\frac{\Delta\mathcal{E}}{\mathcal{T}}\right) \qquad (7)$$

where $\Delta\mathcal{E} = \mathcal{E}(x') - \mathcal{E}(x)$ is the accuracy change between the new solution x', derived from optimized prompt $\hat{\mathcal{P}}_{t+1}$, and the current solution x, generated with original prompt $\mathcal{P}_t$. Here, the temperature $\mathcal{T}$ will gradually decrease with each update. As $\mathcal{T}$ reduces, the probability of accepting suboptimal solutions decreases, enabling the model to escape local optima in the early stages and converge to better solutions later. Building on this, TSA ensures stable and robust accuracy convergence.

In Fig. 4 (b), it is shown that while TLR methods outperform the Naive approach in BBH, they still exhibit substantial variability. To address this issue, we integrate TSA into the TLR approach. The results indicate that the incorporation of TSA into TLR significantly improves both performances and robustness, surpassing the two original baselines.

> **Takeaway:** *Limiting the magnitude, updated parameter size, and direction of each update step to some extent can enhance the robustness and stability of the PO.*

5.2 Exploration for Efficiency

Through our definitions of TLR, TDO, and TSA, we have largely mitigated the issue of update stability. Next, we shift our focus to convergence speed. To enhance convergence efficiency, we propose three methods: Textual Learning Rate Decay (TLRD), Textual Momentum (TMnt) and Textual Contrastive Learning (TCL)

5.2.1 Textual Learning Rate Decay (TLRD)

Inspired by the role of learning rate decay in deep learning, where larger updates facilitate exploration in early stages and smaller updates ensure fine-

tuning later, we define TLRD to enhance convergence speed and stability in prompt optimization. At the beginning of the update process, a higher learning rate encourages substantial modifications to the initial prompt, enabling diverse exploration of potential solutions. As the optimization progresses and the prompt approaches a satisfactory level, the learning rate decreases dynamically to focus on fine-grained refinements.

As shown in Fig. 6 (a,b), we compare the fixed TLR ($\mathcal{R} = 1$) with the learning rate decay strategy both on the BigGSM and MGSM. The results demonstrate that the learning rate decay method achieves much faster convergence compared to the fixed learning rate approach.

5.2.2 Textual Momentum (TMnt)

During each update, the model samples batches independently, without incorporating information from previous ones. This can cause notable bias in gradient optimization, particularly when the batch size is small, as batch-to-batch variations can significantly impact the convergence trajectory.

To mitigate this issue, we draw inspiration from classical machine learning techniques and introduce Textual Momentum (TMnt). By incorporating feedback from past updates, our approach allows the optimizer to recognize previous gradient directions and integrate them into the current update, ultimately refining the final gradient direction for more stable optimization. The final ideal gradient direction $\mathcal{G}_{\text{final}}$ is computed as:

$$\mathcal{G}_{\text{final}} = \mathcal{G}_{\text{current}} + \sum_{i=1}^{3} \gamma^i \cdot \mathcal{G}_{\text{past}_i} \qquad (8)$$

where $\mathcal{G}_{\text{current}}$ is the current batch gradient, $\mathcal{G}_{\text{past}_i}$ represents the i-th previous gradient, and γ is a decay factor (e.g., $\gamma = 0.9$) reducing older gradients' influence. In practice, we do not explicitly define certain details, such as the decay coefficient γ. Instead, we only present the last three feedbacks and encourage the optimizer to focus more on updates closer to the current step to balance recent trends without over-relying on outdated information.

As shown in Fig. 7 (a,b), based on our previously proposed TLR method, we compare the pure TLR with TLR+TMnt on the validation sets of BigGSM and MGSM. It is evident that after adding the TMnt method, the convergence speed has improved.
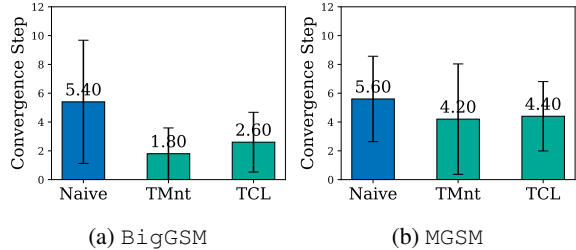

(a) BigGSM   (b) MGSM

Figure 7: We tested the TMnt and TCL methods on the validation set of BigGSM and MGSM environment. The convergence steps and their standard deviations are displayed in the bar chart.

5.2.3 Textual Contrastive Learning (TCL)

Building on the success of contrastive learning in improving model efficiency in DL view (Radford et al., 2021; Qin et al., 2022), we propose Textual Contrastive Learning (TCL), which encourages the model to differentiate high-quality (*positive*) prompts from suboptimal (*negative*) ones, thus enhancing the optimization process.

Specifically, we first evaluate each updated prompt on the training set to obtain its accuracy as a performance metric. Let $\mathcal{A}(x)$ denote the accuracy of prompt x on the training set. Next, we categorize historical prompts into *positive* (higher accuracy) and *negative* (lower accuracy) groups. We then sample one prompt from each group based on an accuracy-weighted probability distribution. If the accuracy difference between the positive and negative prompts falls below a predefined margin, we only encourage the LLM to learn from the positive prompt. Otherwise, we guide the optimizer to: (1) Imitate distinguishing features of positive prompts, represented by the set $\mathcal{F}_+$. (2) Retain shared features between positive and negative prompts, represented by $\mathcal{F}_\cap$. (3) Eliminate undesirable features unique to negative prompts, represented by $\mathcal{F}_-$.

The final gradient update $\mathcal{G}_{\text{final}}$ is computed as:

$$\mathcal{G}_{\text{final}} = \mathcal{G}_{\text{current}} + \mathcal{F}_+ - \mathcal{F}_- \qquad (9)$$

By refining the gradient in this manner, the optimizer learns to reinforce beneficial prompt features while suppressing undesirable ones, ultimately improving the quality of future updates. We provide visualization of the TCL methods in Fig. 8.

To test the effectiveness of TCL, we compare the convergence step of Naive and TCL methods on the validation sets of BigGSM and MGSM. As shown in Figure 7 (a,b), it is evident that after adding the TCL method to naive, the convergence speed have
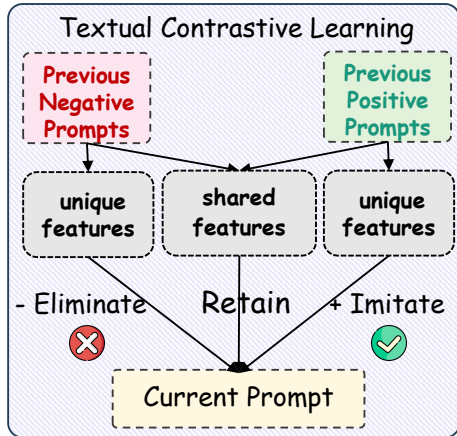
Figure 8: Visualization of the TCL method.

consistently improved. Unfortunately, we also observe that although TMnt appears to outperform the TCL method in the figure, with a faster convergence speed, its final convergence effect is inferior to that of the TCL method and exhibits higher instability.

> **Takeaway:** *Incorporating dynamic, historical, and contrasting directions of gradient optimization from previous update steps can significantly accelerate the convergence speed.*

5.3 Exploration for Generalizablity

Before addressing generalizability, we first propose several methods to enhance model update stability and accelerate convergence. However, our ultimate goal is to improve the model's overall performance, particularly its ability to generalize beyond the training set. Specifically, we aim to extend the learned results to: (1) The validation and test sets of the same dataset. (2) Datasets with similar content but varying levels of difficulty. (3) Datasets involving similar tasks.

Therefore, we train on a subset of the GSM8K training set to derive the final prompt and evaluate its performance on the test sets of GSM8K, BigGSM, BBH, and MATH. Building on the previously proposed effective methods, we adjust the training set size and batch size to examine their impact on the generalizability of our approach. Additionally, we integrate prior techniques and introduce Textual regularization to further explore its role in enhancing generalizability. More experiments on generalization to different LLM models (Gemini (Team et al., 2023), DeepSeek-R1 (Guo et al., 2025)) are presented in the appendix A.4.2.

expname	hyper	GSM8K	BGSM	BBH	MATH
batch size	bs = 3	93.3	**57.3**	79.7	**73.9**
	bs = 6	93.2	52.7	**85.0**	65.6
	bs = 9	93.3	57.2	77.0	70.2
	bs = 12	**93.7**	54.8	79.7	69.6
	bs = 15	89.7	48.4	80.0	70.8
training set size	ts = 50	93.2	52.7	85.0	65.6
	ts = 100	**94.3**	**57.0**	**90.3**	**69.7**

Table 2: Accuracy comparison across different batch sizes and training set sizes. All results are the average of three different seeds. The **bold** is the best while the red is the worst. More details are shown in the Table 9.

5.3.1 Exploration for Training Set Size

Inspired by the training scaling experience in DL (Kaplan et al., 2020), we manage to scale up the training set size to improve the generalization. As shown in Table 2, we create two training subsets by sampling 50 and 100 instances from GSM8K. The model trained on 100 instances consistently outperforms the one trained on 50 across all test sets. This aligns with intuition, as a larger training set offers broader sample coverage, reducing distribution shift and improving generalizability.

5.3.2 Exploration for Batch Size

To explore the impact of different batch sizes on generalizability, we evaluate five batch sizes in terms of final test accuracy, as shown in Table 2. The results indicate that: (1) Within a certain range, a smaller batch size can achieve better performance, but increasing the batch size has a negligible effect on performance. (2) However, if the batch size is excessively large, resulting in too few update steps, the optimization process may fail to converge to the optimal solution.

It is worth noting that the total number of loss calculations remains constant across different batch sizes, while the number of backward passes and updates varies. Therefore, when the requirements for final performance are not overly stringent, it is advisable to appropriately increase the batch size. This reduces the frequency of backward passes and updates, thereby saving computational resources, such as token usage, without significantly compromising model performance.

5.3.3 Textual Regularization (TRegu)

Ockham's razor in DL suggests that simpler models with fewer parameters often generalize better. Building on this principle, we investigate Textual Regularization (TRegu) techniques to reduce prompt complexity and enhance generalization.

expname	GSM8K	BGSM	BBH	MATH
w/o TRegu	93.3	55.0	81.0	67.9
w/ TRegu	**94.0**	**55.9**	**87.2**	**70.5**

Table 3: We compare the performance with and without the addition of TRegu on the baseline methods of TSA, TLRD, and TCL. The final results are obtained by averaging the outcomes across the six seeds. The **Bold** data indicates the highest value in each column.

Specifically, textual L2 regularization is applied to reduce prompt complexity, while L1 regularization is used to promote sparsity by eliminating irrelevant features. Practically, textual L2 regularization encourages simplifying individual sentences, whereas L1 regularization helps remove irrelevant ones. As shown in Table 3, TRegu methods demonstrate significant improvements in generalization on various tasks.

> **Takeaway:** *A larger training dataset and a more precise and streamlined language structure contribute to enhancing the generalizability of the resulting prompt.*

6 Best Practices

By systematically integrating various optimization methods, we progressively improve update robustness, generalization, and efficiency. However, identifying the optimal combination of these techniques remains an open challenge. To address this, we conduct extensive experiments on BigGSM, MGSM, and BBH, evaluating different method combinations to determine the most effective approach. Our findings indicate that the optimal configuration includes TLRD, TSA, TCL, and TREGU, collectively referred to as DLPO. To assess its effectiveness, we compared our approach against several baselines: (1) TextGrad (TG) (Yuksekgonul et al., 2024), a gradient-based prompt optimization method. (2) APO (Pryzant et al., 2023), a widely recognized automatic prompt optimization technique. (3) HUMAN, a human-optimized prompt method derived from the BigGSM dataset.

As shown in Table 4, our method demonstrates state-of-the-art performance on the test sets of both the training datasets and external datasets. It consistently outperforms other methods, including TG and APO, which exhibit lower accuracy across most datasets. This indicates that our method introduces more effective optimization strategies for enhancing prompt quality. Even when compared

Trainset : BigGSM				
Method	BGSM	BBH	MATH	GSM8K
TG	55.7	87.3	69.7	92.7
APO	<u>58.4</u>	<u>89.0</u>	59.7	**93.4**
DLPO	**60.2**	**89.7**	<u>71.3</u>	<u>93.3</u>
HUMAN	54.4	89.0	**72.0**	90.3
Trainset : MGSM				
Method	MGSM	BBH	MATH	BGSM
TG	61.8	77.7	<u>68.8</u>	32.5
APO	<u>82.7</u>	<u>86.0</u>	51.0	<u>47.9</u>
DLPO	**86.7**	**89.3**	**70.8**	**55.6**
Trainset : BBH object counting				
Method	BBH	GSM8K	MATH	BGSM
TG	63.8	83.3	**72.2**	44.8
APO	<u>90.9</u>	<u>92.8</u>	68.9	<u>55.8</u>
DLPO	**94.2**	**93.9**	<u>71.9</u>	**56.7**

Table 4: We test various methods on the trainset of BigGSM, MGSM, and BBH. Except for the HUMAN method, where the prompt is taken directly from the original article, each result represents the average of three different seeds for the corresponding method. The **bold** data indicates the highest value in each column, while the <u>underlined</u> data represents the second-highest value. The complete table is in the appendix 7.

to human-optimized prompts, our approach still achieves superior results.

7 Related Work

Early prompt optimization methods relied on internal access to LLMs (Lester et al., 2021; Li and Liang, 2021; Qin and Eisner, 2021; Gu et al., 2021; Liu et al., 2021). However, these methods often require access to the logits or internal states of LLMs, which is infeasible for those only accessible through APIs (Hou et al., 2023; Tang et al., 2024). To overcome these limitations, recent research has shifted toward exploring gradient-free methods, such as reinforcement learning based methods (Deng et al., 2022; Zhang et al., 2022; Diao et al., 2022), search-based methods (Prasad et al., 2022; Pryzant et al., 2023) and other techniques like evolutionary algorithms (Sun et al., 2022) and boosting (Hou et al., 2023). Among them, Reflection-based methods have gained significant attention due to their ability to iteratively refine prompts while maintaining interpretability. Several notable frameworks exemplify this paradigm, such as ProTeGi(APO) (Pryzant et al., 2023), TextGrad (Yuksekgonul et al., 2024), PromptAgent (Wang et al., 2023), PE2 (Ye et al., 2023), and GPO (Tang et al.,

2024). Apart from the above methods, researchers have also explored key challenges in LLM-based prompt optimization.

Li et al. (2023) revealed that existing prompt optimization methods are vulnerable to distribution shifts, a critical issue for real-world applications. Ma et al. (2024) conducted a comprehensive analysis of LLM-based prompt optimization, introducing a novel method that directly optimizes the target model's behavior in a more controllable manner.

These advances in automated prompt optimization align with our research direction. However, as noted in our problem statement, significant challenges remain in achieving robust, efficient, and generalizable prompt optimization, limiting broader application. We are the first to systematically address these issues, introducing innovative techniques inspired by traditional deep learning to overcome these challenges.

8 Conclusion

Through empirical analysis, we identify key limitations in current reflection-based prompt optimization methods, particularly in robustness, efficiency, and generalization. To address these issues, we introduce novel techniques inspired by traditional deep learning paradigms, integrating concepts like gradient modulation, regularization, and adaptive learning into text-based optimization. Our approach enhances stability, convergence speed, and generalizability in a systematic manner. Experiments on diverse tasks and datasets, including BBH, GSM8K, BigGSM, MATH, and MGSM, validate the effectiveness of our methods. By addressing core challenges in prompt optimization, our work provides valuable insights and guides future research in this field.

Limitations

Although we have introduced multiple methods and demonstrated their effectiveness, a deeper exploration of the intrinsic properties and potential variants of each method is still lacking. Even for the approaches we have developed, there remains considerable room for enhancement in future research. Additionally, we aspire to see our method applied more broadly, extending beyond prompt optimization to a wider range of fields.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiaqi Wang, Mengkang Hu, Zhi Chen, Wanxiang Che, and Ting Liu. 2025. Ecm: A unified electronic circuit model for explaining the emergence of in-context learning and chain-of-thought in large language model. *arXiv preprint arXiv:2502.03325*.

Qiguang Chen, Libo Qin, Jiaqi Wang, Jinxuan Zhou, and Wanxiang Che. 2024. Unlocking the capabilities of thought: A reasoning boundary framework to quantify and optimize chain-of-thought. *arXiv preprint arXiv:2410.05695*.

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. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*.

Shizhe Diao, Zhichao Huang, Ruijia Xu, Xuechun Li, Yong Lin, Xiao Zhou, and Tong Zhang. 2022. Black-box prompt learning for pre-trained language models. *arXiv preprint arXiv:2201.08531*.

Yingjun Du, Wenfang Sun, and Cees GM Snoek. 2024. Ipo: Interpretable prompt optimization for vision-language models. *arXiv preprint arXiv:2410.15397*.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2021. Ppt: Pre-trained prompt tuning for few-shot learning. *arXiv preprint arXiv:2109.04332*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Bairu Hou, Joe O'connor, Jacob Andreas, Shiyu Chang, and Yang Zhang. 2023. Promptboosting: Black-box text classification with ten forward passes. In *International Conference on Machine Learning*, pages 13309–13324. PMLR.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Moxin Li, Wenjie Wang, Fuli Feng, Yixin Cao, Jizhi Zhang, and Tat-Seng Chua. 2023. Robust prompt optimization for large language models against distribution shifts. *arXiv preprint arXiv:2305.13954*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Zekun Li, Baolin Peng, Pengcheng He, and Xifeng Yan. 2024. Evaluating the instruction-following robustness of large language models to prompt injection. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 557–568.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*.

Oscar Mañas, Pietro Astolfi, Melissa Hall, Candace Ross, Jack Urbanek, Adina Williams, Aishwarya Agrawal, Adriana Romero-Soriano, and Michal Drozdzal. 2024. Improving text-to-image consistency via automatic prompt optimization. *arXiv preprint arXiv:2403.17804*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2022. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying lms with mixtures of soft prompts. *arXiv preprint arXiv:2104.06599*.

Libo Qin, Qiguang Chen, Xiachong Feng, Yang Wu, Yongheng Zhang, Yinghui Li, Min Li, Wanxiang Che, and Philip S Yu. 2024a. Large language models meet nlp: A survey. *arXiv preprint arXiv:2405.12819*.

Libo Qin, Qiguang Chen, Fuxuan Wei, Shijue Huang, and Wanxiang Che. 2023. Cross-lingual prompting: Improving zero-shot chain-of-thought reasoning across languages. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2695–2709, Singapore. Association for Computational Linguistics.

Libo Qin, Qiguang Chen, Tianbao Xie, Qixin Li, Jian-Guang Lou, Wanxiang Che, and Min-Yen Kan. 2022. GL-CLeF: A global–local contrastive learning framework for cross-lingual spoken language understanding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2677–2686, Dublin, Ireland. Association for Computational Linguistics.

Libo Qin, Qiguang Chen, Yuhang Zhou, Zhi Chen, Yinghui Li, Lizi Liao, Min Li, Wanxiang Che, and Philip S Yu. 2024b. Multilingual large language model: A survey of resources, taxonomy and frontiers. *arXiv preprint arXiv:2404.04925*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2022. Language models are multilingual chain-of-thought reasoners. *Preprint*, arXiv:2210.03057.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pages 20841–20855. PMLR.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny

Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Xinyu Tang, Xiaolei Wang, Wayne Xin Zhao, Siyuan Lu, Yaliang Li, and Ji-Rong Wen. 2024. Unleashing the potential of large language models as prompt optimizers: An analogical analysis with gradient-based model optimizers. *arXiv preprint arXiv:2402.17564*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2023. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*.

Qinyuan Ye, Maxamed Axmed, Reid Pryzant, and Fereshte Khani. 2023. Prompt engineering a prompt engineer. *arXiv preprint arXiv:2311.05661*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*.

JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why johnny can't prompt: how non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–21.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. 2022. Tempera: Test-time prompting via reinforcement learning. *arXiv preprint arXiv:2211.11890*.

Yongheng Zhang, Qiguang Chen, Min Li, Wanxiang Che, and Libo Qin. 2024. Autocap: Towards automatic cross-lingual alignment planning for zero-shot chain-of-thought. *arXiv preprint arXiv:2406.13940*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. 2024. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue Zhang, Neil Zhenqiang Gong, et al. 2023. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv e-prints*, pages arXiv–2306.

Ziyu Zhuang, Qiguang Chen, Longxuan Ma, Mingda Li, Yi Han, Yushan Qian, Haopeng Bai, Zixian Feng, Weinan Zhang, and Ting Liu. 2023. Through the lens of core competency: Survey on evaluation of large language models. *arXiv preprint arXiv:2308.07902*.

Appendix

A Detailed and Additional Experiments

A.1 Additional Ablation Studies for TLR, TDO and TSA

We conduct comprehensive ablation studies on the BBH Object Counting dataset (using DeepSeek-R1 as the optimizer model and GPT-4-o-mini as the inference model), which demonstrate the following results:

Table 5: Performance comparison on BBH Object Counting

Method	BBH Test Accuracy (%)
TextGrad (TG)	70.4 ± 25.6
TG w/ TLR	84.9 ± 17.6
TG w/ TDO	84.2 ± 6.0
TG w/ TLR+TDO	90.2 ± 3.1
TG w/ TLR+TDO+TSA	$\mathbf{94.3 \pm 2.1}$

These results confirm that both methods significantly improve performance compared to baselines. The combined TLR+TDO method achieves a 19.8% absolute accuracy improvement over the baseline TG. Each method contributes unique stabilization benefits, as evidenced by the reduced standard deviation in the results.

A.2 Additional Ablation Studies for TLR Hyperparameter

We conduct ablation studies on the TLR hyperparameter. Using DeepSeek-R1 as the optimizer model and GPT-4-mini as the inference model, we evaluated three TLR values on the BBH object counting task (each with three random seeds). The constrained TLR (value=1) shows a **14.5 percentage point improvement** over the no-TLR baseline (84.9 ± 17.6 vs 70.4 ± 25.6). As shown in Table 6, increasing the TLR parameter progressively reduces its stabilization effect:

Table 6: TLR Hyperparameter Ablation Studies

TLR Value	Accuracy (%)
1	$\mathbf{84.9 \pm 17.6}$
2	80.1 ± 20.0
4	68.7 ± 29.2
0 (no TLR)	70.4 ± 25.6

A.3 Detailed Experiments in Best Practices

We use GPT-4o as the backward engine and GPT-4o-mini as the forward engine. The models are trained on the BigGSM training set and tested on the BigGSM, BBH, MATH, and GSM8K test sets. The detailed results, including standard deviations, are presented below.

Trainset : BigGSM				
Method	BGSM	BBH	MATH	GSM8K
DLPO	**60.2±2.1**	**89.7±1.5**	71.3±4.4	93.3±3.4
TG	55.7±2.5	87.3±3.7	69.7±3.5	92.7±2.1
APO	58.4±0.4	89.0±3.4	59.7±9.0	**93.4±0.4**
HUMAN	54.4	89.0	**72.0**	90.3
Trainset : MGSM				
Method	MGSM	BBH	MATH	BGSM
DLPO	**86.7±0.2**	**89.3±3.5**	70.8±2.5	**55.6±1.2**
TG	61.8±17.6	77.7±12.0	68.8±3.9	32.5±13.3
APO	82.7±3.4	86.0±9.9	51.0±25.9	47.9±12.2
Trainset : BBH object counting				
Method	BBH	GSM8K	MATH	BGSM
DLPO	**94.2±2.0**	**93.9±0.5**	71.9±1.0	**56.7±0.2**
TG	63.8±2.4	83.3±12.0	**72.2±2.6**	44.8±13.6
APO	90.9±2.3	92.8±1.9	68.9±2.6	55.8±0.7

Table 7: We test various methods on the trainset of BigGSM, MGSM and BBH. Except for the HUMAN method, where the prompt is taken directly from the original article, each result represents the average and the standard deviation of three different seeds for the corresponding method. The **bold** data indicates the highest value in each column.

We conduct additional benchmarks to further validate the effectiveness of the DLPO method. Specifically, we apply DLPO to two types of scenarios: 1.**Knowledge-intensive tasks**: High School History and Formal Logic from MMLU. 2.**Few-shot tasks**: 20-sample scenario from BigGSM.

As shown in Table 8, DLPO demonstrates superior performance compared to the baseline in all cases:

Table 8: Performance comparison of DLPO across different benchmarks

Benchmark	DLPO Score (%)	Baseline (TG) (%)	Human Optimized (%)
MMLU: High School History	88.8	88.3	–
MMLU: Formal Logic	68.1	67.2	–
Few-shot BigGSM (20 samples)	55.5	54.7	54.4

A.4 Detailed and Additional Experiments in Generalizablity

A.4.1 Detailed Experiments in Batch Size and Training set Size

We use GSM8K as the prompt training set and test the performance of different batch sizes and training set sizes on the test sets of GSM8K, BigBSM, BBH, and MATH. The detailed results with standard deviations are as follows:

expname	hyper	GSM8K	BGSM	BBH	MATH
batch size	bs = 3	93.3±1.0	**57.3±1.2**	79.7±12.7	**73.9±3.1**
	bs = 6	93.2±2.3	52.7±4.2	**85.0±13.9**	65.6±5.7
	bs = 9	93.3±1.6	57.2±0.3	77.0±8.7	70.2±1.6
	bs = 12	**93.7±1.5**	54.8±1.6	79.7±13.6	69.6±1.8
	bs = 15	89.7±7.5	48.4±12.4	80.0±12.3	70.8±2.5
trainset size	size = 50	93.2±2.3	52.7±4.2	85.0±13.9	65.6±5.7
	size = 100	**94.3±0.5**	**57.0±0.8**	**90.3±0.6**	**69.7±3.7**

Table 9: Accuracy comparison across different batch sizes and training set sizes. For the batch size experiments, we utilize the TSA, TLRD, and TCL methods, which have shown promising results in previous approaches. Under the benchmark of a training set size of 50, we conduct experiments with batch sizes of 3, 6, 9, 12, and 15. For the training set size experiments, we fix the batch size at 6 and employ the TSA, TLRD, and TCL methods to compare the performance with training set sizes of 50 and 100. All experiments are run with three different seeds, and the mean and standard deviation are calculated from the results. The **bold** data indicates the highest value in each column. The red data indicates the lowest value in each column.

A.4.2 Additional Experiments in Generalizablity of Different LLM

We also test whether the optimized prompts obtained from specific forward (inference LLM models) and backward engines (optimizer LLM models) can generalize to other forward engines. Typically, different forward engines possess varying capabilities and excel in different domains, leading to distinct errors (loss) encountered during reasoning. This, in turn, results in different feedback (gradients), ultimately yielding varied prompts. This presents a significant challenge for generalizing optimized prompts to different forward engine.

We use the optimized prompt results obtained from training on BigGSM, with GPT-4o-mini as the forward engine and GPT-4o as the backward engine, as the system prompt for other forward LLM engines. We test these prompts in the BigGSM environment. The results for GPT-4o and GEMINI-2.0-flash are presented in table 10. The results demonstrate the potential of our DLPO method to generalize to other different forward engines.

Method	GPT-4o-mini (original)	GPT-4o	Gemini-2.0-flash
DLPO	**60.2±2.1**	**77.9±1.1**	**84.0±2.7**
TG	55.7±2.5	75.6±3.4	83.4±1.7
APO	58.4±0.4	75.5±0.2	83.5±1.6
HUMAN	54.4	62.2	80.5

Table 10: We test the optimized results of DLPO, TG, APO of specific backward engine and forward engine on the trainset of BigGSM. Except for the HUMAN method, where the prompt is taken directly from the original article, each result represents the average and the standard deviation of two different optimized prompts of the corresponding method. The **bold** data indicates the highest value in each column.

To explore the capabilities of the latest reasoning models, such as DeepSeek-R1, as the backward engine, we test three methods. The results in Table 11 show that our approach can effectively use DeepSeek-R1 as the backward engine and achieves the best performance among the three methods.

Method	DeepSeek-R1
DLPO	**61.2**
TG	49.8
APO	58.0

Table 11: We test the DeepSeek-R1 as the backward engine for DLPO, TG and APO methods. The **bold** data indicates the highest value in each column.

Although DeepSeek-R1 demonstrates strong performance in optimizing prompts, it requires extended thinking time to generate feedback. Moreover, the improvement it offers over GPT-4o as the optimizer is relatively modest (1.66%). We believe this is likely because GPT-4o has already approached the optimization limit of the prompt for the forward engine (GPT-4o-mini). Therefore, there is currently no compelling need to adopt R1-like models in practice. We plan to conduct further investigations into this in the future.

B Implementation Details and Visualization schematics of the method

B.1 Implementation details of Textual Learning Rate

To address excessively large single-step updates in text optimization, we introduce the **Textual Learning Rate (TLR)** mechanism. This mechanism operates at the sentence level and controls the maximum number of update units allowed in a single optimization step. Each update unit corresponds to one of the following operations: (1).Adding a sentence. (2).Deleting a sentence. (3).Modifying a sentence. The value of TLR, denoted as $\mathcal{R}$, represents the maximum number of update units allowed in a single step. If the LLM intends to make more changes than $\mathcal{R}$, we encourage it to retain impactful changes and discard non-critical ones.

We provide a brief visualization of the TLR methods in Figure 9 right. Below is the original prompt that describes the concept and usage of TLR for LLM prompt optimizer:

> Instruction:
> You need to update the original variable on a sentence level, and the number of updates (including adding sentence, deleting sentence, and modifying sentence) should be limited to a specific quantity (which we call the 'learning rate').
> If the learning rate is: 4, here's an example:
> Initial:
> <VARIABLE>
> As a Math Calculator, please solve:
> Required Steps:
> 1. Identify problem type
> 2. Show calculation steps
> Output Format:
> - Process:
> - Final Result:
> - Verification:
> </VARIABLE>
> Modified Version with exactly 4 changes:
> <IMPROVED-VARIABLE>
> As a reasoning Engine, please solve: [modifying sentence]
> Required Steps:
> 1. Identify problem type
> 2. Show calculation steps
> 3. Analyze complexity [adding sentence]
> 4. Assess stability [adding sentence]
> Output Format:
> - Process:
> - Final Result: [deleting sentence 'Verification:']
> </IMPROVED-VARIABLE>
> Conclusion:
> 1(modify) + 2(add) + 1(delete) = 4. Your learning rate is: $\mathcal{R}$. For each optimize step, please make $\mathcal{R}$ update(s) to the original sentences and keep the other unchanged.

B.2 Implementation details of Textual Dropout

To address the high randomness of LLM updates, we introduce **Textual Dropout (TDO)**. TDO requires the optimizer to randomly "drop" (skip updating) a portion of sentences during each update. This mechanism mitigates the risk of deleting or altering sentences with positive impact and encourages LLM to focus on updating the remaining sentences.

Formally, for $\mathcal{S}$ total sentences and dropout rate p, the number of preserved sentences $\mathcal{K}$ is:

$$\mathcal{K} = \lceil p \cdot \mathcal{S} \rceil \tag{10}$$

The final set of sentences $\mathcal{P}_{\text{new}}$ is:

$$\mathcal{P}_{\text{new}} = \mathcal{P}_{\text{preserved}} + \mathcal{P}_{\text{updated}},$$

where $\mathcal{P}_{\text{preserved}}$ are the preserved sentences, and $\mathcal{P}_{\text{updated}}$ are the updated sentences based on gradient text $\mathcal{G}_{\text{text}}$. We provide a brief visualization of the TDO methods in Figure 9 left.

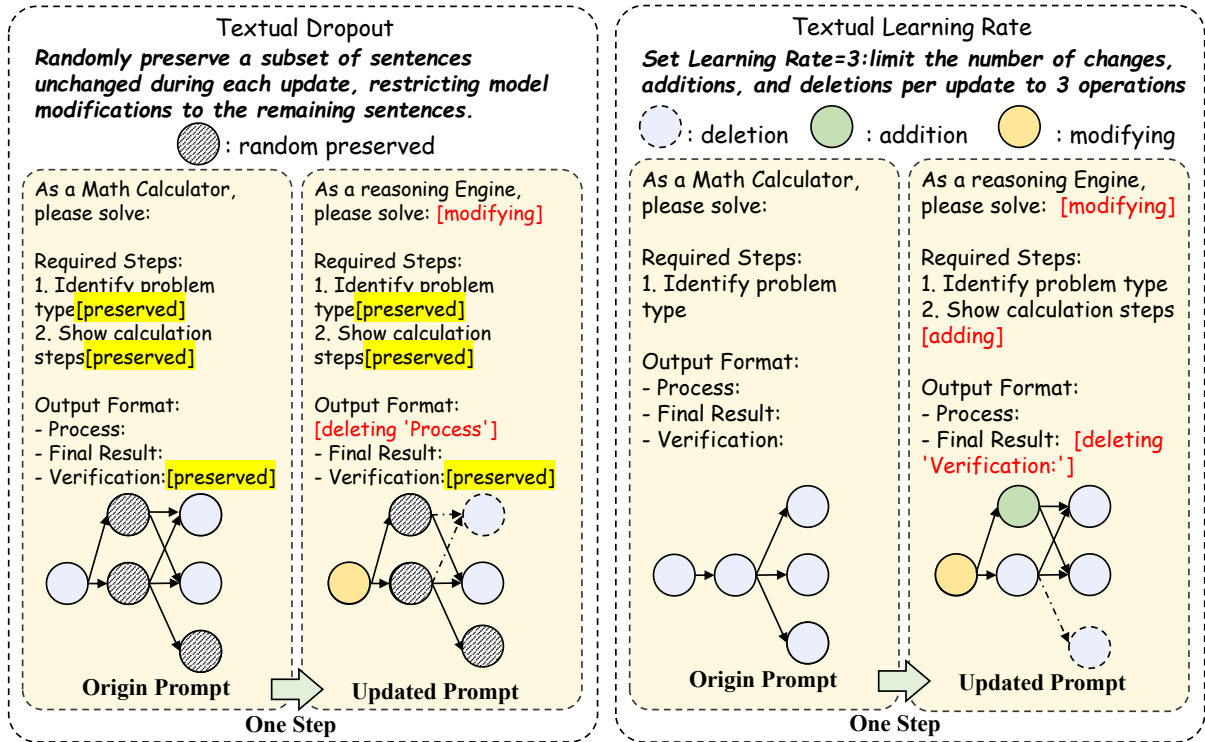Below is the algorithm design for implementing TDO:

Figure 9: Visualization of the TDO (left) and TLR (right) methods

---

Algorithm 1 Textual Dropout Algorithm

Require: Original text T_{original}, Dropout rate p
Ensure: Modified text T_{new}
  1: Split T_{original} into sentences $\mathcal{S}$ using punctuation marks (e.g., ., !, ?)
  2: Compute the number of preserved sentences: $\mathcal{K} = \lceil p \cdot |\mathcal{S}| \rceil$
  3: Randomly select $\mathcal{K}$ sentences from $\mathcal{S}$ as $\mathcal{P}_{\text{preserved}}$
  4: Generate a TDO instruction using $\mathcal{P}_{\text{preserved}}$
  5: **return** TDO Instruction

The selected sentences are passed to the LLM optimizer instruction, which ensures that these sentences remain unchanged during the current optimization step. The prompt is formatted as follows:

> Instruction:
> We have introduced a dropout mechanism. The <DROPOUT>'sentences'</DROPOUT> in the original variable need to remain unchanged for this optimize step. You should focus on altering the other sentences.

Here, `sentences` is replaced with the randomly selected sentences from the TDO process.

B.3 Implementation details of Textual Simulated Annealing

Our **Textual Simulated Annealing (TSA)** scheme dynamically combines simulated annealing principles with text optimization, allowing the prompt to escape local optima and converge to better solutions. Below is the pseudocode for implementing TSA:

Algorithm 2 Textual Simulated Annealing Algorithm

Require: initial prompt x_0, initial temperature $\mathcal{T}_0$, cooling rate α, maximum iterations N

1: initialize $x \leftarrow x_0$, $\mathcal{T} \leftarrow \mathcal{T}_0$
2: **for** $i \leftarrow 1$ to N **do**
3:      generate a new solution x' by perturbing x
4:      compute accuracy difference: $\Delta\mathcal{E} = \mathcal{E}(x') - \mathcal{E}(x)$
5:      **if** $\Delta\mathcal{E} \geq 0$ **then**
6:          accept the new solution: $x \leftarrow x'$
7:      **else**
8:          compute acceptance probability: $P = \exp\left(\frac{\Delta\mathcal{E}}{\mathcal{T}}\right)$
9:          generate a random number $r \in [0, 1]$
10:         **if** $r < P$ **then**
11:             accept the worse solution: $x \leftarrow x'$
12:         **end if**
13:     **end if**
14:     update temperature: $\mathcal{T} \leftarrow \alpha \cdot \mathcal{T}$
15: **end for**
16: return x

Brief Explanation

The algorithm starts with an initial set of prompts and gradually adjusts them based on training set accuracy. The temperature parameter $\mathcal{T}$ controls the probability of accepting worse solutions, which decreases over time. This allows the model to explore the solution space early and converge to better solutions later. We provide a brief visualization of the TSA methods in Figure 10.
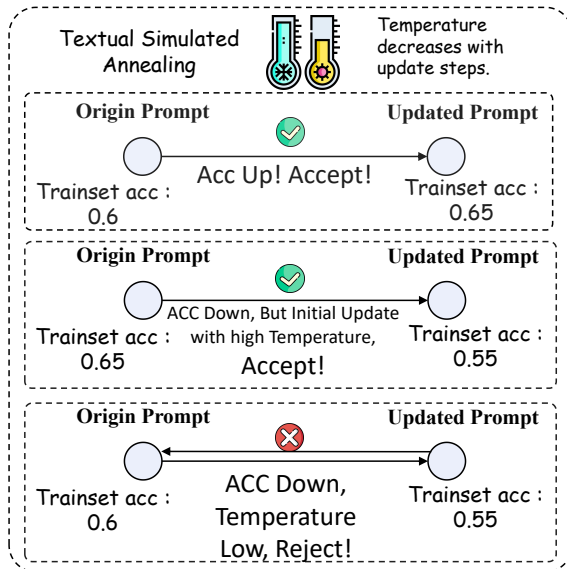
B.4 Implementation details of Textual Learning Rate Decay

Figure 10: Visualization of the TSA methods to refinement throughout the optimization process.

Since we emphasize the exploration of different prompting approaches at the beginning of the update process, it is crucial to encourage the LLM to make substantial modifications to the initial prompt. As the updates progress and the prompt reaches a satisfactory level, the focus shifts to fine-grained refinements, necessitating a lower learning rate for more stable and precise improvements. To achieve this, the learning rate `lr` decreases by one after each update step. For small learning rates ($\text{lr} \leq 4$), the system appends the instruction B.1 to the prompt, enabling fine-tuning with smaller adjustments. Conversely, for large learning rates ($\text{lr} > 10$), the system appends the instruction B.4, which encourages more substantial modifications to explore diverse solutions. This dynamic adjustment ensures a smooth transition from exploration to refinement throughout the optimization process.

> Instruction:
> In order to break away from convention, discover more creative solutions, and explore limitless possibilities, please boldly unleash your imagination based on feedback and make transformative modifications to the previous variables. Do not be confined by existing forms; courageously break the mold and experiment with entirely new combinations and ideas, as this may spark unexpected and groundbreaking outcomes.

B.5 Implementation Details of Textual Momentum

To address the issue of batch-to-batch variability in optimization, we introduce **Textual Momentum (TMnt)**, which leverages feedback from past updates to stabilize and refine the optimization process. During each update, the optimizer considers historical feedback from the last three iterations, presented using the following instruction:

> Instruction:
> Here is the historical feedback on this variable: <PAST-FEEDBACK>history</PAST-FEEDBACK>
> Please analyze the main trends and patterns in the feedback across different iterations. If the feedback consistently points to similar issues or suggests insufficient modifications, it indicates that the changes made to the variable are not substantial enough. The later history feedback will be more accurate and relevant. In such cases, please propose more significant and impactful adjustments to the variable to better address the feedback and improve its performance.

This instruction encourages the backward engine to analyze trends in historical feedback and propose meaningful adjustments when necessary. The final gradient direction is computed by incorporating past gradients with a decay factor γ, as follows:

$$\mathcal{G}_{\text{final}} = \mathcal{G}_{\text{current}} + \sum_{i=1}^{3} \gamma^i \cdot \mathcal{G}_{\text{past}_i}, \tag{11}$$

where $\mathcal{G}_{\text{current}}$ is the gradient computed from the current batch, $\mathcal{G}_{\text{past}_i}$ represents the gradient from the i-th previous iteration, and γ is a decay factor (e.g., $\gamma = 0.9$) that reduces the influence of older gradients. To balance recent trends without over-relying on outdated information, only the last three feedbacks are considered, and the decay factor ensures that older feedback has diminishing influence. In practice, the optimizer dynamically adjusts its reliance on past feedback based on the consistency and relevance of the trends observed in the historical data, ensuring that the optimization process is both stable and adaptive.

B.6 Implementation Details of Textual Contrastive Learning

To refine the optimization process, we introduce **Textual Contrastive Learning (TCL)**, which encourages the backward engine to distinguish high-quality (positive) prompts from low-quality (negative) ones. The historical prompts are split into two groups: the first half represents positive prompts, while the second half represents negative prompts. For positive prompts, a decreasing weight scheme prioritizes earlier high-quality samples, whereas for negative prompts, an increasing weight scheme emphasizes more recent low-quality samples. If a valid negative prompt is found (accuracy difference exceeds a predefined threshold), both positive and negative prompts are included in the final template; otherwise, only the positive prompt is used. The final gradient update is computed as:

$$\mathcal{G}_{\text{final}} = \mathcal{G}_{\text{current}} + \mathcal{F}_+ - \mathcal{F}_-, \tag{12}$$

where $\mathcal{F}_+$ represents beneficial features from positive prompts, and $\mathcal{F}_-$ represents undesirable features from negative prompts.

Below is the pseudocode for implementing TCL:

Algorithm 3 Textual Contrastive Learning Algorithm (TCL)

Require: Historical Prompts ($\mathcal{H}$), Accuracy Threshold (τ)
Ensure: Updated Prompt Template
 1: Split $\mathcal{H}$ into Positive Prompts ($\mathcal{P}$) and Negative Prompts ($\mathcal{N}$).
 2: Assign decreasing weights to $\mathcal{P}$ and increasing weights to $\mathcal{N}$.
 3: Sample two prompts from $\mathcal{P}$ and two from $\mathcal{N}$, using their respective weights.
 4: **for** each sampled $n \in \mathcal{N}$ **do**
 5:    **if** $\max(\mathcal{P}.\text{accuracy}) - n.\text{accuracy} \geq \tau$ **then**
 6:       Mark n as valid.
 7:    **end if**
 8: **end for**
 9: **if** valid Negative Prompts exist **then**
10:    Generate a instruction using both $\mathcal{P}$ and valid $\mathcal{N}$.
11: **else**
12:    Generate a instruction using only $\mathcal{P}$.
13: **end if**
14: **return** Generated Textual Contrastive Learning Instruction
---

The following is the instruction that uses both Positive and valid Negative Instruction for contrastive learning.

> Instruction:
> You can learn valuable insights by comparing the good and bad variables from past data. On the training set, the better-performing variables are <Positive-VAR>$\mathcal{P}$</Positive-VAR>, while the poorer-performing variables are <Negative-VAR>$\mathcal{N}$</Negative-VAR>. To improve your variable, focus on adopting the unique features that contribute to the success of the better variables and eliminate the unique features associated with the poorer variables. This approach will help enhance performance and avoid repeating past mistakes.

The following is the instruction that only encourages imitation of positive prompts.

> Instruction:
> You can gain valuable insights by analyzing high-performing variables from historical data. On the training set, the top-performing variables are «Positive-VAR>$\mathcal{P}$</Positive-VAR>.

It is worth noting that if we use both TSA and TCL, the solutions rejected by TSA are actually tested for accuracy on the train set, and as a result, we will also include this result in our historical prompts for TCL.

B.7 Implementation Details of Textual Regularization

To enhance the generalization capability of our approach, we incorporate regularization techniques into the text optimization process. Specifically, we apply L2 regularization to simplify overly complex sentences and L1 regularization to eliminate irrelevant ones. In the context of text gradients, each sentence is treated as a distinct feature.

The implementation involves the following steps:

L2 Regularization (Simplification): We encourage simplifying individual sentences by reducing their complexity while preserving their core meaning. This is achieved through the following instruction:

> Instruction:
> Please simplify the overly complex and lengthy sentences in the variable. Ensure the output concise, easy to understand, and suitable for a general audience.

L1 Regularization (Sparsity): We promote sparsity by removing sentences that are deemed irrelevant or detrimental to the overall meaning. The corresponding instruction is as follows:

> Instruction:
> If you are certain that a particular sentence in the variable has no impact on the overall meaning or purpose or has a negative effect, please delete that sentence. However, if you believe that all sentences are useful and contribute to the overall meaning, then retain all sentences. Ensure that the final variable maintains clarity, coherence, and relevance.

This approach ensures that the optimized text is both concise and meaningful, striking a balance between simplicity and relevance.

C Experimental Setup

Apart from the additional experiments provided in the appendix A.4.2, all experiments in the main text consistently use GPT-4o as the Backward engine and GPT-4o-mini as the Forward engine.

C.1 Benchmark Description

In our experiments, we utilize the below benchmarks to evaluate the effectiveness of our proposed methods:

- GSM8K: A collection of 8.5K high-quality grade school math word problems designed to assess language models' multi-step mathematical reasoning capabilities, which are the most widely used for primary-school-level mathematical reasoning (Cobbe et al., 2021).

- MATH: A dataset covering a wide range of topics to test advanced mathematical reasoning, which are another widely used for hard college-level mathematical reasoning (Hendrycks et al., 2021).

- BigGSM: An extension of the GSM8K dataset, including more complex and diverse grade school math problems to further challenge model performance, which utilzed to gradually extreme data detection on LLMs (Chen et al., 2024, 2025).

- BigBenchHard (BBH): A subset of the BIG-Bench benchmark focusing on particularly challenging tasks across various domains, designed to push the limits of language model understanding and reasoning.

- MGSM: A multilingual version of the GSM8K dataset, containing grade school math problems translated into multiple languages to evaluate models' cross-lingual mathematical reasoning abilities, which are widely utilized for multilingual scenario mathematical evaluation (Qin et al., 2024b, 2023; Zhang et al., 2024).

C.2 Experimental Setup in Exploration for TMnt and TCL

In the Exploration for Efficiency section, we employ Textual Contrastive Learning and Textual Momentum methods to expedite convergence speed. We define convergence as the validation set accuracy of the prompt reaching a certain threshold and maintaining this level for three consecutive update steps (the threshold is set at 0.5 for BigGSM and 0.8 for MGSM). We showcase the average and standard deviation of the convergence steps using five different seeds.

C.3 Experimental Setup in Best Practices Section

In the context of best practices, our DLPO, classical TG, and APO methods are assessed on the training datasets of BigGSM, MGSM, and BBH. The final step results are chosen for evaluation. With the exception of the HUMAN technique, which relied on prompts extracted directly from the source articles, each reported performance metric reflects the average of three distinct seed values per method. The hyperparameters utilize across the three distinct training datasets are detailed in Table 12.

Argument	BigGSM	MGSM	BBH
Trainset Size	200	50	50
Backward Engine	GPT-4o	GPT-4o	GPT-4o
Forward Engine	GPT-4o-mini	GPT-4o-mini	GPT-4o-mini
Batch Size	3	2	3
Epochs	1	1	2
T_{LR}	False	False	False
T_{DO}	False	False	False
T_{Regu}	True	True	True
T_{CL}	True	True	True
T_{SA}	True	True	True
T_{Mnt}	False	False	False
T_{LRD}	60	25	30

Table 12: Hyperparameter settings for different trainsets in Best Practices section

D   Optimized Results on Benchmark Datasets

In this section, we present the three original prompts before averaging in the best practices section, each generated with the same hyperparameters but different random seeds. Additionally, the updated results are influenced by the uncertainty of the API interface. Each prompt represents the last prompt obtained throughout the entire updating process. It is worth noting that our prompt will be sent into the API as a system prompt, rather than directly concatenated with the question.

D.1   Results on BigGSM Dataset

Optimized Prompt 1:
Provide a clear, detailed step-by-step calculation for the reasoning question. Emphasize accuracy by verifying calculations at each step, especially in multiplication, and reassessing both intermediate and final results. Outline all assumptions and define variables explicitly to reduce ambiguity. Ensure transparency by showing intermediate steps, enabling error detection and understanding. After obtaining the solution, cross-check and simplify calculations for consistency. Evaluate the plausibility of results in a real-world context to identify any anomalies. Present the final numerical answer clearly, with consistent units where applicable. Remain cautious of overconfidence and express uncertainty if appropriate.

Optimized Prompt 2:
For reasoning questions, provide clear, detailed, and contextually relevant responses. To ensure numerical accuracy and clarity:
1. **Context Recognition**: Define the scenario and identify key parameters. Use suitable formulas and units for the context.
2. **Calculation Transparency**: Clearly show each calculation step and verify accuracy with known results or alternative methods.
3. **Error Checks**: Detect and correct errors in real-time.
4. **Format Flexibility**: Use diverse examples to demonstrate adaptability to different numerical formats, like with and without commas.
5. **Consistent Formatting**: Use standard numerical presentations, such as "1,234," for readability unless specified otherwise.
6. **Format Validation**: Cross-check numerical formats with trusted sources for compliance.
7. **Feedback Use**: Continuously refine formats using feedback to enhance precision.
8. **Final Check**: Ensure final outputs meet expected formats and make adjustments as needed.
Final Output: Present the final number clearly on a new line, typically formatted as "9876" for easy identification.

Optimized Prompt 3:
Deliver precise and effective answers to reasoning questions by implementing these strategies:
1. **Clarify the Question:** Restate the question to ensure complete understanding.
2. **Simplify Calculations:** Break down arithmetic tasks into smaller steps and verify each.
3. **Log Steps Clearly:** Document each calculation step methodically, minimizing rounding errors until the final result.
4. **Verify Results:** Cross-check numerical results using alternative methods or sources.
5. **Use Examples:** Apply relatable examples for better understanding.
6. **Restate Instructions:** Clearly articulate all numerical relationships to avoid omissions.
7. **Detect Errors:** Use checks for inconsistencies, such as unexpected numerical values, to identify errors.
8. **Intermediate Checks:** Implement sanity checkpoints or intermediate results for large calculations.
9. **Iterate and Improve:** Continuously refine responses through feedback and iterative checks.
10. **Prioritize Complexity:** Address the most complex or error-prone aspects first.
These strategies focus on accurate data extraction, direct calculations, contextual interpretation, and consistent cross-verification, enhancing clarity, precision, and adaptability in reasoning tasks.

D.2 Results on MGSM Dataset

Optimized Prompt 1:
1. Clearly state the main answer in a simple numeric form: "Answer: [numeric value]".
2. Use the following strategies for clarity and precision:
- Utilize bullet points for concise step-by-step guidance.
- Include only the necessary context for calculations.
- Prioritize numerical precision and verify accuracy.
- Define variable roles consistently and standardly.
- Implement checks to ensure alignment with expected results and avoid common errors.
- Manage units and contexts effectively.
- Conclude with a concise summary of key supporting points.

Optimized Prompt 2:
Break down the reasoning process into clear steps, ensuring correct unit handling. Verify unit consistency and apply necessary conversions. Use a structured format to improve clarity.
1.**Explicit Assumptions**: Clearly state any assumptions, especially those related to units or context. Make these explicit to avoid misunderstandings.
2.**Mathematical Verification**: Verify all calculations and unit conversions explicitly. Check each step for accuracy, using past examples for cross-reference.
3.**Diverse Reasoning Methods**: Apply both deductive and analogical reasoning. Ensure the purpose of each step is clear, including unit handling.
4.**Clear Language**: Use precise language to avoid ambiguities, especially regarding units and quantities. Make each step understandable.
5.**Intermediate Steps**: Break down reasoning into intermediate steps to show logical flow and the significance of each part.
6.**Feedback Anticipation**: Anticipate potential feedback or errors, especially with unit conversions. Adapt approaches to address these proactively.
7.**Confidence Levels**: Express uncertainties and confidence levels, focusing on conclusions related to unit handling.
8.**Verified Final Answer**: Provide a precise final answer, checking for accuracy in unit-related aspects. Highlight the final answer for clarity.
9.**Exact Matching and Clarification**: Ensure outputs match the correct answers exactly and include contextual clarifications, focusing on units.
10.**Self-Evaluation**: Evaluate responses for unit accuracy, clarity, and brevity. Address key points thoroughly.
Maintain consistent response quality by integrating feedback to enhance precision and relevance. Use adaptive learning to focus on historical errors and improve unit handling continuously.

Optimized Prompt 3:
1. Provide a concise numerical answer to the reasoning question through these steps:
- Analyze the query to identify essential data points directly relevant to the answer.
- Use a step-by-step logical computation approach, clearly linking each step to the query's context.
- Consistently use terminology from the input data to prevent ambiguity. - Verify calculations with provided data, providing brief justifications for each key operation.
- Maintain a consistent numerical format to prevent parsing errors and adhere to local and cultural conventions if specified.
- Tailor your explanation to the user's expertise: Offer detailed steps for beginners and concise steps for advanced users.
- Present the final numerical answer in square brackets, e.g., [42], in its simplest form, ensuring consistent units unless additional formatting is specified.
- Guarantee the output is a single, precise numerical value, fulfilling evaluation metrics and ensuring logical coherence.

D.3 Results on BBH Object Counting Trainset

Optimized Prompt 1:
You will address quantitative reasoning and arithmetic questions with precision and clarity. Ensure arithmetic accuracy by verifying calculations and cross-referencing results with known benchmarks. Break down calculations into clear steps for transparency and unambiguous answers. Contextualize responses by considering relevant prior information.
Implement robust error detection and handling, including recalculations and user clarification requests when discrepancies arise. Verify counts explicitly by listing items and confirming totals. Use clear language to confirm counts and details, highlighting important numerical information.
Adopt a strategy of self-evaluation, focusing on accuracy and relevance before finalizing responses. Avoid unnecessary elaboration unless requested. Develop a systematic approach for identifying and correcting errors, including a secondary verification mechanism for counts.
Deliver concise responses to boost clarity and user satisfaction, especially when detailed explanations are unnecessary. Address ambiguous queries by clarifying assumptions or soliciting additional information. Ensure uniform formatting to emphasize critical numerical data. Continuously adapt responses to enhance user comprehension.
Establish a feedback loop to learn from past interactions, refining response strategies to align with expected outcomes. Use digit form for numerical data to align with ground truth expectations. Format responses for easy parsing by automated systems, ensuring consistency with the ground truth. Implement a mechanism for continuous improvement of numerical accuracy based on feedback.
Use examples from training data to guide response formatting and accuracy. Ensure responses logically follow from the query and context, directly addressing the question. Incorporate a mechanism for detecting and correcting errors by flagging inconsistencies and prompting user verification. Use examples that demonstrate clear enumeration and counting strategies to align responses with best practices.

Optimized Prompt 2:
You will answer reasoning questions with a focus on tasks that may involve counting. Follow these streamlined guidelines to ensure clarity, precision, and efficiency:
1. **Task Recognition**: Identify if the question involves counting and provide a precise numerical answer.
2. **Direct Instructions**: Recognize lists or enumerations as signals for counting tasks and deliver a straightforward numerical count.
3. **Contextual Awareness**: Be aware of contextual cues that suggest a counting task and ensure your response is relevant.
4. **Logical Response Structure**: Clearly list relevant items, then provide a total count, using simple formatting to distinguish steps.
5. **Verification and Accuracy**: Verify your numerical responses for accuracy and consistency with the listed items.
6. **Example Utilization**: Use concise examples to illustrate clarity and consistency in handling counting tasks.
7. **Efficiency Feedback Loop**: Internally verify your responses to maintain accuracy and logical flow.
By following these strategies, you will deliver concise, precise, and contextually appropriate answers to reasoning questions, optimizing for runtime efficiency.

Optimized Prompt 3:
Begin each response by restating the task briefly for clarity. Provide the main answer upfront, especially for numerical queries, and include explanations only if they add significant value. Use bullet points or lists for complex information to enhance clarity. Avoid redundancy and unnecessary details, ensuring responses are concise and self-contained. Address ambiguities by seeking clarification or clearly stating assumptions. Maintain a conversational tone and anticipate follow-up questions to improve interaction. Present reasoning in logical steps and offer detailed breakdowns of calculations when necessary. Implement verification mechanisms, such as checklists or summary statements, to confirm accuracy. Ensure numerical operations align with expected results and conclude with a verification statement. Consistently align responses with the original query, rephrasing if needed for comprehension. Learn from past inaccuracies to improve future responses. Consider edge cases like zero quantities or duplicates and adjust calculations accordingly. Use a feedback loop for clarification, prompting follow-up questions or stating assumptions if the task is unclear. Integrate numerical examples and encourage step-by-step calculations to ensure consistency and verify results by checking individual counts against the total.