Prompt Optimization for Large Language Models with Generalized Feedback via Semantic Centrality

Anonymous Submission

Abstract

Large Language Models (LLMs) perform well across NLP tasks, but remain sensitive to prompt formulation. Iterative feedback-based prompt optimization methods enhance reasoning, but they often overfit to specific cues, causing biases and poor generalization. We propose a simple, yet effective method that enables a more robust use of feedback-based methods by providing generalized feedback from intermediate steps. By rephrasing and masking overly specific content, our method[1] identifies semantically central concepts that guide prompt construction in a more stable and transferable manner. Experiments on nine tasks demonstrate that our method consistently outperforms CoT prompting, feedback-based optimization strategies, and recent state-of-the-art baselines. Further analysis reveals that generalizing intermediate feedback leads to faster convergence, lower cost, and improved generalization when transferred to different LLMs.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable performance across natural language processing (NLP) tasks (Sahoo et al., 2024; Brown et al., 2020; Bubeck et al., 2023), yet their outputs remain highly sensitive to prompt formulation (Chen et al., 2023; Zamfirescu-Pereira et al., 2023). A prominent advancement, Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022), improves multi-step reasoning by decomposing problems into intermediate steps. Recently, dynamic prompt-generation methods have been explored, leveraging LLMs to iteratively explore and refine their own reasoning steps as an optimization problem (Wang et al., 2023; Yang et al., 2023; Juneja et al., 2024). These iterative loops of feedback and selection have been shown to further boost accuracy on NLP benchmarks. However, recent findings suggest that LLMs can overfit to such intermediate reasoning traces, leading to performance degradation when those steps encode spurious or overly specific information (Saba, 2023; Chu et al., 2024). This issue is particularly pronounced in feedback-based methods (Wang et al., 2023; Yang et al., 2023), where intermediate feedback (steps) are generated from model errors and tend to encode overly specific information. This observation motivates a key research question: *how can we preserve the strengths of intermediate-feedback-driven methods while preventing the accumulation of localized biases that arise from overfitting to*

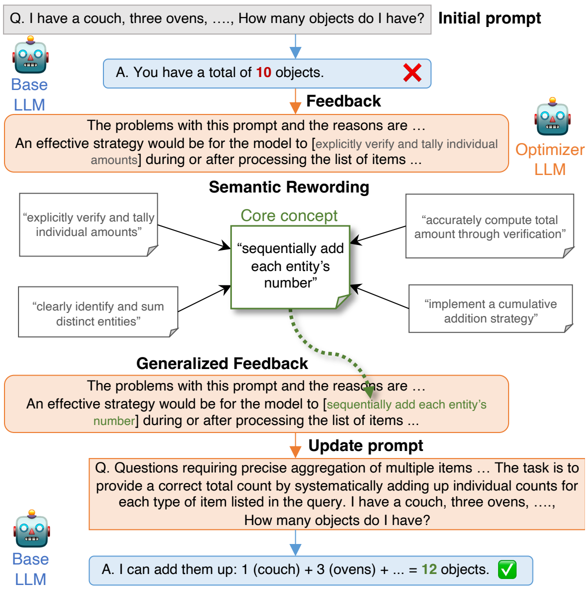

Figure 1: Overview of the proposed method of semantic centrality selection for optimizing LLM prompts. The base LLM's incorrect response from the initial prompt is evaluated by an optimizer LLM, which provides feedback (top). The masking is applied to specific information, generating multiple semantic candidates. The most broadly applicable concept is selected, leading to a more generalized feedback statement. Finally, the prompt is updated to reflect the generalized feedback (bottom).

[1]Our code will be provided at `https://example.com`.

overly specific information in intermediate steps?

We address this by proposing a novel prompt optimization method that generates intermediate feedback with sufficient semantic abstraction or generalization. Rather than relying directly on potentially biased sentences, our method rewords overly specific content, and identifies the core semantic concepts that remain stable across variations. These concepts are then used to guide prompt construction in the base LLM, leading to stronger generalization.

We validate our method on a diverse set of nine tasks, comparing against human prompts, CoT prompting, feedback-based strategies, and recent state-of-the-art baselines. We further investigate how intermediate feedback affects accuracy, convergence, cost, and transferability of optimized prompts to other LLM.

2 Feedback Generalization

Our method is based on error-feedback-based planning with MCTS, similar to PromptAgent. We utilize MCTS, iteratively searching for prompts through selection, expansion, simulation, and backpropagation. This prompt-optimization strategy treats the current prompt as a current state and explores the next state through an action that searches for improved prompts using error feedback from incorrect answers. In selection, we follow the UCT algorithm (Kocsis and Szepesvári, 2006) to balance exploration and exploitation. In expansion, the optimizer LLM generates feedback and new different prompts (Wang et al., 2023). These undergo simulation, where a reward function evaluates their effectiveness, and the results are propagated backward in back-propagation to prioritize high-reward paths.

2.1 Semantic Rewording

Algorithm 1 shows the action process of the proposed method. Our action process begins with a specialized prompt template $\mathcal{T}$, which incorporates the initial prompt $\mathcal{P}$. Specifically, this template guides the optimizer LLM $\mathcal{O}$ to identify exactly $\mathcal{M}$ segments, $\mathcal{C}$, within $\mathcal{P}$ that are too specific in the domain. Each identified segment $c_i \in \mathcal{C}$ is then replaced with a unique placeholder token, denoted as $[\mathrm{MASK}_i]$ for $i = 1, \ldots, \mathcal{M}$, and for each masked segment, the model generates precisely $\mathcal{K}$ alternative candidate replacements, collectively represented as $\mathcal{C}'_i = \{c'_1, c'_2, \cdots, c'_{\mathcal{K}}\} \subseteq \mathcal{C}'$.

Algorithm 1 Semantic Centrality Selection

Require: Prompt template: $\mathcal{T}$, Optimizer LLM: $\mathcal{O}$, Embedding model: g

Generate $(\mathcal{F}, \mathcal{C}, \mathcal{C}') \leftarrow \mathcal{O}(\mathcal{T})$
$\mathcal{S} \leftarrow \varnothing$
for $i \leftarrow 1$ to $\mathcal{M}$ **do**
  for all $c'_k \in \mathcal{C}'_i$ **do**
    $s \leftarrow \mathcal{F}$
    $s \leftarrow$ replace c_i in s with c'_k
    $\mathcal{S} \leftarrow \mathcal{S} \cup s$
  end for
end for
Compute $\mathbf{E} \leftarrow g(\mathcal{S})$
Compute $I(s) = \sum_{s' \in \mathcal{S}} \langle \mathbf{e}_s, \mathbf{e}_{s'} \rangle$
$F_g \leftarrow \arg\max_{s \in \mathcal{S}} I(s)$
return F_g

This process can be formalized as a transformation $\mathcal{O} : \mathcal{T} \to (\mathcal{F}, \mathcal{C}, \mathcal{C}')$, where the optimizer LLM $\mathcal{O}$ generates the initial feedback text $\mathcal{F}$ with the prompt template $\mathcal{T}$, masked placeholders $\mathcal{M}$ and the corresponding candidate substitutions $\mathcal{C}'$, allowing a structured and systematic refinement of the original prompt (Madaan et al., 2023). It plays a crucial role in ensuring robustness and adaptability. Rather than relying on a single sentence, the incorporation of diverse alternatives allows for a more broader exploration of possible sentences.

2.2 Semantic Centrality-based Selection

After the candidate generation process, the semantic centrality selection strategy is employed to determine the most general and robust feedback. Let each candidate $s \in \mathcal{S}$ be assigned to a vector embedding $\mathbf{e}_s \in \mathbb{R}^d$ by a text embedding model $g : \mathcal{S} \to \mathbb{R}^d$. The similarity between two candidates s and s' is quantified by the inner product $\langle \mathbf{e}_s, \mathbf{e}_{s'} \rangle$. We then define a score $I(s)$ for each candidate as following equation

$$I(s) = \sum_{s' \in \mathcal{S}} \langle \mathbf{e}_s, \mathbf{e}_{s'} \rangle. \quad (1)$$

This score reflects the degree to which a given candidate represents the overall set in the embedding space. In practice, we compute the embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{S}| \times d}$, where each row corresponds to an embedding $\mathbf{e}_s$, and then obtain $I(s)$ by adding the rows of the matrix product $\mathbf{E}\mathbf{E}^\top$. The candidate feedback texts are ranked based on their scores, and we select the top-1 index as generalized feedback $\mathcal{F}_g$, marginalizing the others.

Method	BIG-Bench tasks							General NLU tasks				Total Avg.
	Penguins	Geometry	Epistemic	Obj. Count.	Temporal	Causal Judge.	Avg.	Subj	TREC	CB	Avg.	
Human (ZS)	98.73	48.00	84.40	78.80	91.00	64.00	77.49	66.80	65.60	80.36	70.92	75.30
Human (FS)	97.47	41.50	80.60	49.40	93.40	58.00	70.06	85.30	75.20	85.71	82.07	74.06
CoT (ZS)	97.47	55.50	82.00	93.20	90.80	65.00	80.66	67.80	65.00	**87.50**	73.43	78.25
CoT	97.47	69.50	87.40	93.80	96.80	66.00	85.16	79.60	77.40	85.71	80.90	83.74
OPRO	**100.00**	83.36	82.15	88.23	97.93	66.94	86.44	73.42	80.62	74.40	76.15	83.01
PromptAgent	98.73	77.50	**89.20**	94.80	**99.60**	67.00	87.81	73.90	85.40	82.14	80.48	85.36
UniPrompt	**100.00**	78.90	81.75	93.50	99.44	**69.89**	87.11	**88.19**	82.34	86.28	**85.60**	86.61
Ours	98.73	**85.00**	**89.20**	**96.20**	98.60	69.00	**89.46**	80.00	**86.00**	82.14	82.71	**87.21**

Table 1: Comparison across BBH tasks and General NLU tasks. Test accuracy (%) is reported.

The score $I(s)$ quantifies how central a candidate is within the overall distribution in the embedding space. Since semantically similar embeddings cluster together (Elekes et al., 2017), this score tends to prioritize candidates near the centroid of the distribution. A previous work (Saba, 2023) suggests that such central representations often correspond to higher-level abstractions, making them more generalizable. Consequently, this method reduces the reliance on domain-specific expressions and favors feedback that is more broadly applicable across diverse contexts.

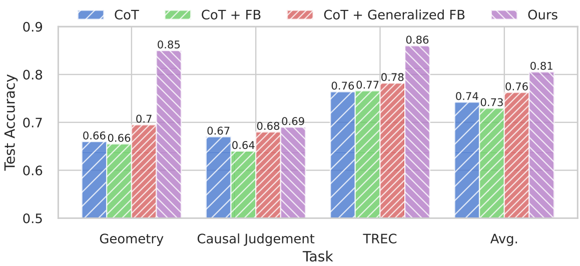

Figure 2: Comparison with different feedback strategies, highlighting the impact of our generalization method.

3 Experiments

3.1 Set-up

Datasets. The datasets are used to evaluate the effectiveness of prompt optimization on various reasoning and classification tasks from a subset of challenging BIG-Bench tasks (Suzgun et al., 2023; Srivastava et al., 2023) (Penguins in a table, Object Counting, Epistemic Reasoning, and Temporal Sequences) and general natural language understanding (NLU) tasks, i.e., Subjective (Pang and Lee, 2004), TREC (Voorhees and Tice, 2000), and CB (De Marneffe et al., 2019). Each dataset is divided into train, validation, and test sets. More detailed description is reported in Appendix D.

Baselines. For baseline comparisons, we consider both human-crafted and automated prompt optimization techniques. The human prompt (ZS) and the few-shot version of human prompt (FS) (Suzgun et al., 2023) baselines provide a reference for manually designed prompts, while CoT and zero-shot version of CoT (ZS) serve as strong reasoning-based baselines. Meanwhile, we compare our method against strong baselines including OPRO (Yang et al., 2023), PromptAgent (Wang et al., 2023), UniPrompt (Juneja et al., 2024). We provide the implementation details in Appendix C.

3.2 Comparative Accuracy

Table 1 compares the proposed method with several baselines, including human baselines, CoT baselines, OPRO, PromptAgent, and UniPrompt, on both BBH and General NLU tasks. Specifically, on the six BBH tasks, our method achieves the highest average accuracy of 89.46%, surpassing UniPrompt's 87.11%. While our method does not surpass the strong baseline UniPrompt in the average score across the three general NLU tasks (85.60), it achieves the second-best performance with an average of 82.71, outperforming PromptAgent (80.40). Notably, when evaluated across all tasks, our method attains the highest overall average score of 87.21. These results demonstrate that incorporating more generalized feedback to the base LLM can enhance both the specialized problem-solving abilities and the broader language understanding skills of LLM.

3.3 Feedback Generalization Analysis

Figure 2 shows the effects of error feedback on model performance in CoT settings, focusing on the overfitting issue. While incorporating error feedback (CoT+FB) leads to overfitting and reduces performance, generalized feedback (CoT+Generalized FB) improves generalization, demonstrating significant performance boosts. The proposed method, which combines generalized feedback with MCTS, further enhances perfor-

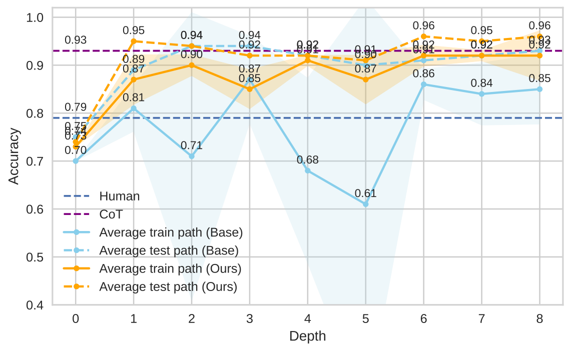

Figure 3: Convergence analysis of different methods based on tree depth in the task of Object Counting. Shaded regions indicate the variance.

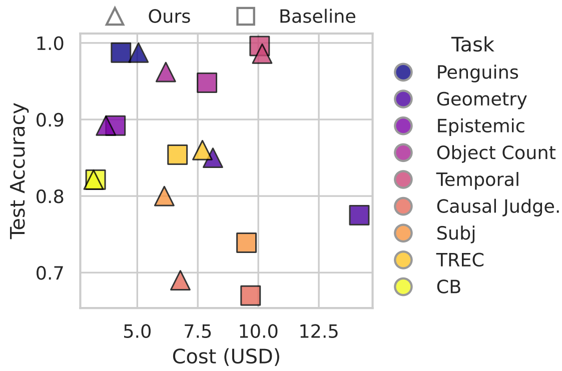

Figure 4: Cost analysis of our method (triangles) and the baseline (squares) in both cost and test accuracy. Each task is represented by different colors.

mance, leading to more consistent results. More results and details are reported in Appendix J.

3.4 Convergence Analysis

Figure 3 shows the performance trends in varying tree depths for Human, CoT, PromptAgent (Base), and the proposed method (Ours) in the task of Object Counting. The trajectories illustrate the evolution of average performance during both training (reward) and testing steps. As tree depth increases, ours consistently outperforms the base model, demonstrating superior generalization by effectively narrowing the gap between training and testing performance. Notably, ours not only achieves higher performance after full exploration, but also shows a better result over Human and CoT baselines even at earlier depths.

3.5 Optimization Cost Analysis

Figure 4 compares our method with the baseline (PromptAgent) in terms of cost and test accuracy in various tasks. Despite incorporating an additional embedding model, ours achieves a lower average cost (6.34 USD vs 7.74 USD) while achieving a higher test accuracy, as reported in Table 1. This suggests that our method not only improves accuracy, but also reduces search costs more effectively. Furthermore, it indicates that our generalization has ability to condense contexts (Zhou et al., 2022).

3.6 Prompt Generalization Analysis

To assess the transferability of prompts across base LLMs, we evaluate whether prompts optimized on GPT-4o-mini retain their effectiveness when applied to a other base LLM, GPT-o3 mini in Table 2. Notably, we observe that the optimized prompts maintain high performance across most tasks even when transferred to GPT-o3 mini. While other methods tend to degrade on the Causal Judge benchmark, our method exhibits improved performance. This indicates that generalized feedback not only enhances performance but also yields prompts that are robust to model shifts, highlighting the practical utility of our method.

Task	GPT-4o mini (source)			GPT-o3 mini (target)		
	PromptAgent	UniPrompt	Ours	PromptAgent	UniPrompt	Ours
Penguins	98.73	**100.00**	98.73	**100.00**	**100.00**	98.73
Geometry	77.50	78.90	**85.00**	97.50	93.50	**98.50**
Epistemic	**89.20**	81.75	**89.20**	88.40	87.20	**92.60**
Object.Count.	94.80	93.50	**96.20**	99.40	99.40	**99.60**
Temporal	**99.60**	99.44	98.60	**99.60**	99.20	99.20
Causal Judge.	67.00	**69.89**	69.00	65.00	67.00	**70.00**
Subj	73.90	**89.19**	80.00	74.60	**89.70**	79.50
TREC	85.40	82.34	**86.00**	85.00	77.00	**87.80**
CB	82.14	**86.28**	82.14	85.71	**89.28**	87.50
Avg.	85.36	86.61	**87.21**	88.36	89.14	**90.38**

Table 2: Prompt generalization performance. Prompts optimized on GPT-4o mini demonstrate effective transfer to other stronger base LLM GPT-o3 mini.

4 Concluding Remarks

In this paper, we identify the risk of prompt overfitting in LLMs and emphasize the necessity of prompt generalization to achieve robust prompts. Furthermore, our findings show that properly generalizing the context can effectively mitigate the risk. Building on these findings, we propose a simple method called semantic centrality selection that restructures model feedback to extract general core concepts while reducing reliance on task-specific patterns. Through experiments across nine tasks, our method demonstrates superior generalization while leading to faster convergence, reducing cost, and improving transferability to other LLMs.

Limitations

While effective in the general tasks tested, its performance in specialized domains is still uncertain and may rather decrease. Moreover, our method can be computationally intensive, especially with many candidates. Furthermore, the use of another pre-trained model and additional algorithms raises concerns about increased computational costs. Our future work will focus on balancing generalization with the domain-specific details required in complex reasoning scenarios.

Ethical Statement

We follow fundamental ethical principles to ensure responsible use of datasets while minimizing potential social harms. All datasets are sourced from publicly available materials and used in accordance with applicable privacy and copyright regulations.

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.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. arxiv. *arXiv preprint arXiv:2303.12712*.

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2023. Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735*.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1173–1203.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *Proceedings of Sinn und Bedeutung*, volume 23 of 2, pages 107–124.

Abel Elekes, Martin Schaeler, and Klemens Boehm. 2017. On the various semantics of similarity in word embedding models. In *2017 ACM/IEEE Joint Conference on Digital Libraries*, pages 1–10.

Gurusha Juneja, Nagarajan Natarajan, Hua Li, Jian Jiao, and Amit Sharma. 2024. Task facet learning: A structured approach to prompt optimization. *arXiv preprint arXiv:2406.10504*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Bo Pang and Lillian Lee. 2004. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, page 271–es, USA. Association for Computational Linguistics.

Walid S Saba. 2023. Stochastic llms do not understand language: towards symbolic, explainable and ontologically based llms. In *International Conference on Conceptual Modeling*, pages 3–19. Springer.

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*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Ellen M. Voorhees and Dawn M. Tice. 2000. Building a question answering test collection. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, page 200–207, New York, NY, USA. Association for Computing Machinery.

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*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

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.

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*.

A Reproducibility Statement

In all experiments, we used OpenAI's API and acknowledge that the experimental results may vary depending on the knowledge cutoff of the models provided by OpenAI[2]. The knowledge cutoff[3] for GPT-4o mini and GPT-o3 mini is October 2023, for GPT-4 Turbo is December 2023, and the `text-embedding-3-small` model was released in January 2024. Nevertheless, to ensure reproducibility, we provide our source code for public access, the data split details in Table 3, exact input prompts in Table 4, 6, 7, and 8, and templates in Table 5. In addition, we manually set the seeds to 42 as the dataset loader incorporates some degree of randomness.

B Computing Environment and License

All experiments were conducted on Ubuntu 22.10, with an Intel(R) Xeon(R) Gold 6326 CPU (2.90GHz) and NVIDIA A100 GPUs (80GB). The source code for this study is adapted from PromptAgent (licensed under Apache-2.0). All the datasets used in the experiments are publicly available.

[2]https://openai.com/

[3]https://platform.openai.com/docs/models.

C Implementation Details

In the experiment, we use the same evaluation strategy as PromptAgent, GPT-4o mini as the base LLM, GPT-4 turbo as the optimizer LLM, and the text embedding model g as `text-embedding-3-large` from OpenAI. In all experiments, we set $\mathcal{M}$ to 2 and $\mathcal{K}$ to 50 for the purpose of computational efficiency. We set the temperature to 0.0 for the base LLM and 1.0 for the optimizer LLM. We set the number of iterations for MCTS to 12, adding 3 children of the leaf node to the tree, the maximum depth to 8, the minimum depth to 2, and the exploration weight to 2.5 for UCT Algorithm. We select the test accuracy based on the node with the highest reward, which is calculated on the validation set.

D Dataset Description

We use six tasks from Big-Bench[4] (Srivastava et al., 2023), and three common NLU tasks. The description of each task is as follows.

Task	Train	Val.	Test
Penguins in A Table	70	70	79
Geometry	150	150	200
Epistemic Reasoning	300	200	500
Object Counting	150	150	500
Temporal Sequences	150	150	500
Causal Judgement	90	90	100
Subjective	200	200	1000
TREC	200	200	500
CB	125	125	56

Table 3: The number of samples in the training, validation, and test data split.

Big-Bench tasks. The `Penguins in A Table task` presents a structured reasoning challenge in which models must answer questions based on a tabular representation of penguin attributes. Each row corresponds to an individual penguin, and attributes such as age, height, and weight are provided. It requires the model to extract relevant information from the table, perform numerical reasoning.

The `Geometry` task tests the model's ability to recognize and classify geometric shapes based on their scalable vector graphics (SVG) paths. It contains a variety of simple and complex geometric

[4]https://github.com/google/BIG-bench.

figures, and the task demands an understanding of how sequences of drawing commands correspond to different shapes.

In the `Epistemic Reasoning` task, models must determine whether a given premise entails a hypothesis. It evaluates the model's ability to recognize epistemic distinctions, such as differentiating between beliefs and factual statements. Some statements involve indirect inferences or degrees of uncertainty, making it necessary for the model to reason beyond simple surface-level text matching.

The `Object Counting` task focuses on numerical reasoning by presenting scenarios in which objects are grouped or described in varying ways. The challenge requires the model to correctly count items while accounting for linguistic ambiguities, such as plural forms, implicit quantities, and collectively referenced objects. It is particularly useful for assessing how well the models handle numeracy and aggregation.

The `Temporal Sequences` task assesses the model's ability to understand and reason about chronological events. Given a sequence of activities, the model must infer when a particular event could have taken place. It involves constraints such as available time slots, activity durations, and event dependencies, which requires the model to accurately track and manipulate time-based information.

The `Causal Judgment` task evaluates the model's understanding of causal relationships. Given a scenario, the model must determine whether an event was a direct result of a preceding action or simply correlated. It is particularly challenging, as it requires distinguishing between correlation and causation, as well as recognizing implicit intentions behind actions.

General NLU. The `Subjective`[5] (Pang and Lee, 2004) consists of text classification examples in which the model must determine whether a given sentence is subjective or objective. Subjectivity is typically marked by personal opinions, emotions, or qualitative judgments, whereas objectivity is associated with factual statements. The challenge involves correctly identifying linguistic cues that signal subjective interpretation without being misled by neutral descriptive language.

The `TREC`[6] task (Voorhees and Tice, 2000) is a well-established question classification benchmark. Given a natural language question, the model must categorize it into one of several predefined types, such as entity, abbreviation, location, or numeric value. It tests the model's ability to interpret the intent behind a question and align it with a structured taxonomy of possible responses.

The `CB`[7] task (De Marneffe et al., 2019) is designed for natural language inference, requiring models to determine whether a hypothesis is entailed, contradicted, or neutral concerning a given premise. It contains sentences with varying levels of implicit meaning, requiring careful semantic interpretation to distinguish between direct entailment, contradiction, and unrelated statements.

E Related Work

Chain-of-Thought. Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022) has emerged as a widely used technique to improve the reasoning capabilities of large language models (LLMs) by decomposing complex problems into a sequence of intermediate steps. This structured reasoning process has proven effective in a variety of tasks, including arithmetic, logic, and commonsense inference. While CoT prompting enhances interpretability and multi-hop reasoning, recent studies highlight its vulnerability to overfitting (Saba, 2023; Chu et al., 2024), as LLMs tend to latch onto spurious patterns or overly specific details present in intermediate steps. Such over-reliance can lead to degraded performance when the CoT trace includes noisy or biased content. Furthermore, the deterministic nature of static CoT exemplars limits their flexibility and adaptability across tasks and model variants. Our work builds on these insights by aiming to generalize intermediate reasoning signals to prevent such overfitting and improve the robustness of CoT-style prompting methods.

Automatic prompt optimization. To move beyond handcrafted prompts and improve model performance across diverse tasks, a growing body of work frames prompt optimization as a search or planning problem. Recent approaches have explored ways to leverage LLMs themselves as optimizers, using feedback or task performance as supervision. PromptAgent (Wang et al., 2023) treats prompt optimization as a sequential decision-making problem and applies Monte Carlo Tree

[5] `https://huggingface.co/datasets/SetFit/subj`.
[6] `https://huggingface.co/datasets/CogComp/trec`.
[7] `https://huggingface.co/datasets/aps/super_glue`.

Search (MCTS) to iteratively refine prompts. An optimizer LLM generates feedback based on errors from a base LLM, and each simulation step yields new prompt variants that are scored and prioritized for future rollouts. OPRO (Yang et al., 2023) proposes a simpler iterative approach where LLMs optimize prompts based on meta-prompt trajectories that include previously generated solutions and their corresponding scores. The optimizer LLM uses this history to generate new instructions that incrementally improve performance. UniPrompt (Juneja et al., 2024) attempts to address generalization by structuring prompts into semantically meaningful sections and editing them based on feedback aggregated from mini-batches.

Our method proposes a layer of semantic abstraction within feedback-driven optimization. Rather than applying raw feedback directly, we rephrase and perturb intermediate reasoning signals, extract recurring core concepts, and use these generalized semantics to construct prompts. This mitigates overfitting to error-specific details and stabilizes iterative optimization methods such as MCTS.

F Preliminaries & Background

In this section, we provide detailed preliminaries and background, as it would offer clearer context for understanding the proposed method. Our method builds on error-feedback-driven prompt planning using MCTS, following the general strategy of PromptAgent (Wang et al., 2023). In this setting, prompt optimization is formulated as a sequential decision-making process where each node in the search tree represents a prompt state, and edges correspond to transformation actions derived from feedback. The optimizer LLM acts as a planner that proposes feedback (actions), while the base LLM serves as an environment that evaluates candidate prompts (states) under those actions. MCTS proceeds iteratively with four stages: Selection, Expansion, Simulation, and Backpropagation.

Selection. During this phase, the algorithm traverses the search tree starting from the root node, selecting child nodes based on the Upper Confidence bound applied to Trees (UCT) criterion (Kocsis and Szepesvári, 2006; Wang et al., 2023).

Expansion. If the selected node is not fully expanded, the optimizer LLM generates feedback on the current prompt, identifying weaknesses and suggesting specific revisions. These feedback signals serve as action candidates, which are applied to the current prompt to produce new prompt variants. These variants form child nodes (i.e., expanded states) in the tree.

Simulation. Each new prompt is evaluated by querying the base LLM to obtain a response, which is then scored using a reward function tailored to the task. This function quantifies the utility of the generated prompt-response pair (e.g., accuracy).

Backpropagation. The obtained reward is propagated back through the selected path in the tree, updating statistics such as visit counts and expected reward estimates. This allows the tree policy to increasingly favor prompt transformations that lead to high-performing prompts in future rollouts.

While the MCTS-based method provides a principled approach for navigating the prompt space by iteratively refining prompts based on feedback, it is inherently susceptible to overfitting to narrow, instance-specific cues. This is particularly problematic in the *Expansion* and *Back-propagation* phases, where biased or overly specific intermediate feedback—generated from the optimizer LLM based on the base LLM's error signals—can lead to prompt variants that reinforce spurious reasoning patterns. As the optimization progresses, such biases are amplified across multiple rollouts, resulting in degenerate convergence to prompts that perform well on seen queries but generalize poorly to unseen inputs or alternative LLMs.

Our method addresses this vulnerability by incorporating *semantic centrality-based feedback generalization* directly within the MCTS loop. Specifically, during the Expansion phase, rather than applying feedback as-is, we rephrase feedback signals and identify recurring semantic components that are invariant across variations. These identifiedcore concepts are then used to construct new prompt candidates, effectively regularizing the tree expansion process. This abstraction not only mitigates the risk of localized overfitting, but also improves convergence stability and cross-model transferability. By embedding this generalization mechanism within each iteration of MCTS, our method ensures that the search process remains grounded in semantically meaningful transformations.

G Computational Complexity

We analyze the computational complexity of our prompt optimization method, focusing on the train-

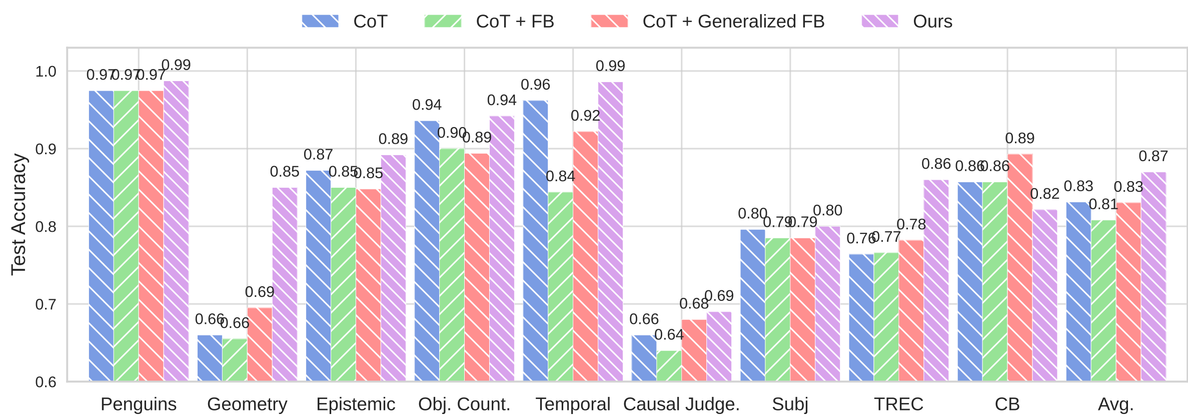

Figure 5: Comparison with different feedback strategies across 9 tasks.

ing phase. Let N denote the mini-batch size and M the complexity of a single model inference. Let $\mathcal{K}$ denote the number of candidate rewording steps per masked segment. In PromptAgent, a candidate prompt is optimized via search and evaluation over training data. Its per-iteration complexity can be approximated as: $\mathcal{O}(NM + M)$ The term NM corresponds to prompt evaluation over a batch of training examples, and the additional M accounts for the optimizer LLM's generation.

In contrast, our method first generates $\mathcal{K}$ candidates, resulting in $\mathcal{K}M$ inference cost from the optimizer LLM. Then, it embeds all candidate prompts using an embedding model and computes a semantic centrality score for each candidate. This re-ranking process involves $\mathcal{O}(\mathcal{K}^2)$ inner product computations and an additional $\mathcal{O}(\mathcal{K})$ for the final selection via $\arg\max$, leading to the total complexity: $\mathcal{O}(NM + M + \mathcal{K}M + \mathcal{K}^2 + \mathcal{K})$ While our complexity scales with $\mathcal{K}$, this dependency is minor in practice because $\mathcal{K} \ll M$; the number of reworded candidates is far smaller than the cost of full model inference over a batch. Under this assumption, the total complexity simplifies to: $\mathcal{O}((N + \mathcal{K})M)$ As shown in Figure 4, our method also produces shorter prompts on average, thereby reducing the overall token-level inference cost despite the additional semantic filtering step.

H Human Prompts

We report the human prompts used in our experiments in Table 4. These prompts serve as the base input for prompt optimization, with additional task-specific prompts—covering task descriptions, questions, and answer formats—incorporated for each sample.

I Feedback Template

The exact template used to generate feedback is reported in Table 5. We create the template considering the planning process with mini-batching. This template facilitates a structured approach to analyzing incorrect examples, identifying specific segments, and generating semantic replacements.

J Details of Feedback Generalization Analysis

Figure 5 investigates the impact of different feedback mechanisms on the performance of the model within a few-shot CoT-based prompt on all datasets. Specifically, we evaluate the risk of overfitting induced by error feedback in models that take advantage of a few-shot CoT. The results are consistent across most tasks. We provide the experimental details as follows. The experimental setup compares several configurations: the baseline few-shot CoT (Table 6), a version with feedback integrated into the training process (CoT+FB, Table 7), a variation employing generalized feedback (CoT+Generalized FB, Table 8), and the proposed method which integrates generalized feedback with MCTS (Ours).

The comparison of Tables 6, 7, and 8 highlights key information on the role of feedback in prompting strategies. The baseline few-shot CoT (Table 6) follows a step-by-step reasoning approach, ensuring interpretability and logical breakdown. However, it lacks explicit feedback to guide error correction, making it susceptible to systematic errors.

Incorporating direct error feedback (CoT+FB, Table 7) improves interpretability by explicitly addressing common mistakes, as seen in the highlighted incorrect answers and explanations. Al-

Task	Human prompt
Penguins in A Table	Answer questions about a table of penguins and their attributes.
Geometry	Name geometric shapes from their SVG paths.
Epistemic	Determine whether one sentence entails the next.
Object Count	Questions that involve enumerating objects of different types and asking the model to count them.
Temporal	Answer questions about which times certain events could have occurred.
Causal Judge.	Answer questions about causal attribution.
Subj	Given the text, choose between 'subjective' and 'objective'.
TREC	Tag the text according to the primary topic of the question. Choose from (A) Abbreviation, (B) Entity, (C) Description and abstract concept, (D) Human being, (E) Location, (F) Numeric value.
CB	Read carefully the following premise and hypothesis, and determine the relationship between them. Choose from 'contradiction', 'neutral' and 'entailment'.

Table 4: Human prompt used in the experiments.

though this provides immediate correction, it risks overfitting the model to specific feedback patterns, as it tends to memorize corrections instead of developing a more generalized problem-solving strategy. This issue is evident in the experimental results, where CoT+FB performs less well compared to the standard CoT, with a lower average accuracy (0.8079 vs. 0.8344). The decline is especially noticeable in tasks requiring flexible reasoning, such as Geometry.

To mitigate this, the CoT+Generalized FB approach (Table 8) introduces a structured feedback mechanism that not only identifies errors but also generalizes the reasoning process by categorizing object types explicitly. This method reinforces filtering and classification skills, improving generalization in various tasks. As a result, it outperforms CoT+FB with an average accuracy of 0.8304, demonstrating a stronger ability to handle tasks that require adaptive problem-solving.

The proposed method, which integrates generalized feedback with MCTS, further enhances this approach by dynamically guiding the reasoning process and mitigating the risk of overfitting. Including MCTS facilitates a more robust exploration of reasoning paths, allowing for a balanced approach between specific error correction and generalization. This results in superior performance across multiple domains, particularly in tasks that require abstraction and long-term dependencies.

In conclusion, the observed performance gains underscore the importance of designing feedback mechanisms that do not anchor the model to specific error cases. Generalized feedback, unlike error-specific feedback, promotes a more adaptable reasoning process, helping the model to improve its performance in tasks that require generalized logical thinking. By combining generalized feedback with MCTS, our method effectively guides the model towards more refined and contextually appropriate solutions, providing a path to more robust and scalable few-shot learning systems.

Section	Description
Purpose	I'm writing prompts for a language model designed to handle various scenarios in a general and robust way. This means the model must identify or construct a solid plan that leads to correct, plan-oriented answers.
Current prompt	Below is my current prompt: {cur_prompt}.
Issues	Despite aiming for a plan-based solution, this prompt fails to address the following examples correctly: {example_string}.
Analysis	Please examine each incorrect example step by step. - Concentrate on how the existing plan (or lack thereof) leads to the wrong answer. - Pay special attention to any deficiencies in how the prompt organizes or outlines steps, rather than focusing on a single domain, version, or specific detail.
Requirements	Then, produce an integrated feedback that addresses these common plan-related issues collectively. Your feedback should highlight any overarching problems in the prompt's plan, propose corrections or improvements to that plan, and ensure that your advice remains sufficiently abstract and broadly applicable—avoid overly specific or domain-constrained details unless absolutely necessary for clarity.
Masking process	Next, identify exactly {mask_num} segments in your feedback that are too narrow, overly technical, or domain-specific. - Replace each identified sentence with a unique [MASK_n] placeholder (e.g. [MASK_1], [MASK_2], ...). - For each [MASK_n], propose {candidate_num} alternative candidates that broaden or generalize the concept, so the final plan remains applicable to a variety of scenarios.
Key reminders	- The goal is to understand the deeper, shared reasons for the planning failures and how to create a more robust plan overall. - The final feedback should be broadly applicable, rather than tailored to a single domain or overly specific detail. - The [MASK_n] replacements and candidate phrases should reflect more generalized or inclusive expressions.
Output formatting	Be sure that in the final feedback: 1. You only have {mask_num} total [MASK_n] tokens. 2. Each token has a different index (e.g., [MASK_1], [MASK_2], ... up to [MASK_{mask_num}]). 3. You provide exactly {candidate_num} candidates for each [MASK_n].
Example	Feedback without [MASK]: [Your feedback in a single consolidated paragraph or set of paragraphs, before inserting any [MASK]]. Feedback with [MASK]: [Your feedback text, but with the identified overly specific sentences replaced by [MASK_n]]. Then, for each mask token, provide candidates in the form: <START>{Candidates_[MASK_1]:[candidate1, ..., candidate_num]}<END>.

Table 5: Template for generating structured error feedback.

Input

Q: I have a blackberry, a clarinet, a nectarine, a plum, a strawberry, a banana, a flute, an orange, and a violin. How many fruits do I have?

A: Let's think step by step.

We first identify the fruits on the list and include their quantity in parentheses:

- blackberry (1)
- nectarine (1)
- plum (1)
- strawberry (1)
- banana (1)
- orange (1)

Now, let's add the numbers in parentheses:

1 + 1 + 1 + 1 + 1 + 1 = 6.

The answer is 6.

Q: I have three oranges, a piano, an apple and two clarinet. How many musical instruments do I have?

Output

We first identify the musical instruments in the list:

- piano (1)
- clarinet (2)

Now, let's add the numbers:

1 + 2 = 3.

The answer is 3.

Table 6: Baseline few-shot chain-of-thought prompting (CoT). This setting provides step-by-step reasoning without error feedback, ensuring clear logic flow. The context, highlighted in blue, structured thought decomposition, emphasizing the identification and categorization of relevant objects.

Input
Q: I have a blackberry, a clarinet, a nectarine, a plum, a strawberry, a banana, a flute, an orange, and a violin. How many fruits do I have? A: The most common wrong answer is 9. The mistake occurs when people count all listed objects instead of **filtering out the non-fruits**. Items such as **"clarinet," "flute," and "violin" are not fruits and should not be included in the total count.** Let's think step by step. We first identify the fruits on the list and include their quantity in parentheses: - blackberry (1) - nectarine (1) - plum (1) - strawberry (1) - banana (1) - orange (1) Now, let's add the numbers in parentheses: 1 + 1 + 1 + 1 + 1 + 1 = 6. The answer is 6. Q: I have three oranges, a piano, an apple and two clarinet. How many musical instruments do I have?
Output
We first **identify the fruits** on the list and include their quantity in parentheses: - orange (3) - apple (1) Now, let's add the numbers in parentheses: 3 + 1 = 4. The answer is 4.

Table 7: Few-shot chain-of-thought prompting with error feedback (CoT+FB). This setting incorporates explicit error identification, as highlighted in red, allowing the model to recognize common mistakes. The cyan-colored context still emphasizes logical reasoning but may lead to overfitting by enforcing rigid feedback structures.

Input
Q: I have a blackberry, a clarinet, a nectarine, a plum, a strawberry, a banana, a flute, an orange, and a violin. How many fruits do I have? A: The most common wrong answer is 9 , the total number of items in the list. Errors occur when people fail to differentiate between relevant and irrelevant objects. It is important to filter out **objects that do not belong to the target category**. Let's think step by step. We first identify the fruits on the list and include their quantity in parentheses: - blackberry (1) - nectarine (1) - plum (1) - strawberry (1) - banana (1) - orange (1) Now, let's add the numbers in parentheses: 1 + 1 + 1 + 1 + 1 + 1 = 6. The answer is 6. Q: I have three oranges, a piano, an apple and two clarinet. How many musical instruments do I have?
Output
We first identify the **objects that belong to the target** on the list and include their quantity in parentheses: - orange (3, not a musical instrument) - piano (1, musical instrument) - apple (1, not a musical instrument) - clarinet (2, musical instruments) Now, let's add the number of musical instruments: 1 (piano) + 2 (clarinets) = 3. The answer is 3.

Table 8: Few-shot chain-of-thought prompting with generalized error feedback (CoT+Generalized FB). This setting refines error feedback by introducing a more structured approach, highlighted in yellow , which guide the model in differentiating between relevant and irrelevant objects. The structured feedback promotes better generalization and mitigates overfitting risks.