Self-Evolving Memory for Automatic Prompt Optimization

Anonymous ACL submission

Abstract

Automatic prompt optimization is a promising approach for adapting large language models (LLMs) to downstream tasks, yet existing methods typically search for a specific prompt specialized to a fixed task. This paradigm limits generalization across heterogeneous queries and prevents models from accumulating reusable prompting knowledge over time. In this paper, we propose **MemAPO**, a memory-driven framework that reconceptualizes prompt optimization as ***generalizable and self-evolving experience accumulation***. MemAPO maintains a dual-memory mechanism that distills successful reasoning trajectories into reusable strategy templates while organizing incorrect generations into structured error patterns that capture recurrent failure modes. Given a new prompt, the framework retrieves both relevant strategies and failure patterns to compose prompts that promote effective reasoning while discouraging known mistakes. Through iterative self-reflection and memory editing, MemAPO continuously updates its memory, enabling prompt optimization to improve over time rather than restarting from scratch for each task. Experiments on diverse benchmarks across both closed and open-ended tasks show that MemAPO consistently outperforms representative prompt optimization baselines while substantially reducing optimization cost. Source code is available at https://anonymous.4open.science/r/memapo.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks, including logical reasoning, mathematical problem solving, and knowledge-intensive question answering (Jaech et al., 2024; Comanici et al., 2025; Yang et al., 2025; Guo et al., 2025; Zeng et al., 2025). In practice, however, the performance of LLMs is highly sensitive to prompt design. Carefully crafted prompts can substantially improve model performance, while poorly specified prompts often lead to unstable or suboptimal outputs (Li and Liang, 2021; Lester et al., 2021; Liu et al., 2024, 2022; Dong et al., 2024). As a result, prompt engineering has become a critical interface for adapting general-purpose LLMs to downstream applications (Wei et al., 2022; Deng et al., 2023; Huang et al., 2026).

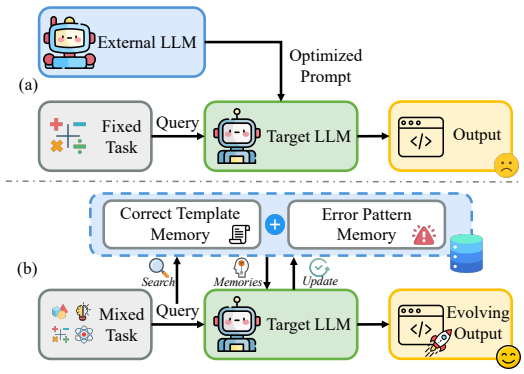

Figure 1: Comparison between MemAPO and existing works. (a) **Existing paradigm**: an external LLM refines prompts for one specific task. (b) **MemAPO**: self-organizes reusable memories across multiple tasks.

Despite its importance, designing effective prompts remains a labor-intensive and iterative process that requires both domain expertise and extensive experimentation. To alleviate this burden, recent studies have explored automatic prompt optimization (APO), which leverages LLMs to iteratively refine prompts based on task feedback (Zhou et al., 2022; Pryzant et al., 2023; Yang et al., 2023; Guo et al., 2023). Existing APO approaches typically treat prompt optimization as a search problem in natural language space. Representative strategies include text-gradient-based optimization (Pryzant et al., 2023; Yuksekgonul et al., 2025), trajectory-based prompt refinement (Yang et al., 2023; Tang et al., 2025), and evolutionary prompt search (Fernando et al., 2023; Guo et al., 2023). These meth-

ods have shown promising improvements in controlled settings by automatically generating and refining prompts through iterative evaluation (Yuksekgonul et al., 2025; Xiang et al., 2025).

However, most existing APO methods are formulated as a prompt search problem, where the objective is to identify a specific prompt that performs well on a fixed task distribution, as illustrated in Figure 1-(a). While effective in controlled settings, this formulation faces fundamental challenges in realistic environments. First, **prompt optimization is inherently unstable due to the extremely large and stochastic prompt space**. Small variations in prompts can lead to significantly different reasoning behaviors, making optimization brittle and often requiring repeated exploration. Second, **useful prompting knowledge is frequently reusable across queries**. Many tasks share underlying reasoning strategies, such as decomposition, verification, or step-by-step deduction. Treating prompt optimization as an isolated search process for each task fails to capture these shared structures and prevents the model from accumulating transferable prompting knowledge. These observations suggest that prompt optimization should not be viewed merely as prompt search. Instead, it is more naturally framed as a *continual knowledge accumulation* process, where reusable prompting strategies are progressively discovered and refined through interaction with tasks. Moreover, **simply compressing historical interactions into a single summary is insufficient**. Successful and failed reasoning trajectories contain fundamentally different types of information: successful cases reveal effective reasoning strategies, while failures expose recurring mistakes that should be avoided. A practical system needs to accumulate and organize both types of experience to guide future reasoning.

Motivated by this perspective, as shown in Figure 1-(b), we propose **MemAPO**, a self-evolving memory framework for automatic prompt optimization agents. MemAPO enables a target LLM to progressively accumulate prompting experiences and reuse them across heterogeneous tasks. Specifically, MemAPO maintains a dual-memory mechanism that captures both successful strategies and recurrent failure signals. Successful prompting trajectories are distilled into reusable reasoning strategies, while incorrect generations are abstracted into structured failure rules that help prevent recurring mistakes. To handle heterogeneous query distributions, MemAPO dynamically retrieves relevant experiences from memory based on the characteristics of each input query. Retrieved reasoning strategies provide guidance for effective problem solving, while failure rules help the model avoid previously observed mistakes when constructing prompts. When existing experiences fail to provide adequate guidance, the model autonomously generates new strategies through a self-reflection mechanism and updates the memory accordingly, enabling the prompting knowledge to evolve over time. Through this process, prompt optimization is transformed *from an externally supervised search procedure into an internal capability that continuously improves through experience accumulation*. We evaluate MemAPO on diverse prompt optimization benchmarks across both closed and open-ended tasks. Experimental results show that MemAPO consistently outperforms representative automatic prompt optimization baselines across multiple datasets and LLM backbones. In addition, by amortizing optimization cost through experience reuse, MemAPO significantly reduces token consumption and API calls, making it more practical for large-scale deployment.

Our contributions are summarized as follows:

- **Problem Formulation**. We introduce a new perspective on prompt optimization by framing it as a self-evolving experience accumulation problem, moving beyond the traditional paradigm of searching for a task-specific optimal prompt.
- **Model Design**. We propose MemAPO, a self-evolving memory framework that enables LLMs to internalize, store, and reuse prompting experiences through a dual-memory mechanism capturing both strategy templates and error patterns.
- **Comprehensive Evaluation**. Extensive experiments across diverse prompt optimization benchmarks demonstrate that MemAPO achieves superior performance and generalization while substantially reducing optimization cost.

2 Related Work

2.1 Automatic Prompt Optimization

Research on engineering effective prompts for LLMs has been rapidly advancing. Early works like chain-of-thoughts (Wei et al., 2022; Kojima et al., 2022), developed through human insights and extensive experimentation, significantly boost the performance of LLMs. To further automate

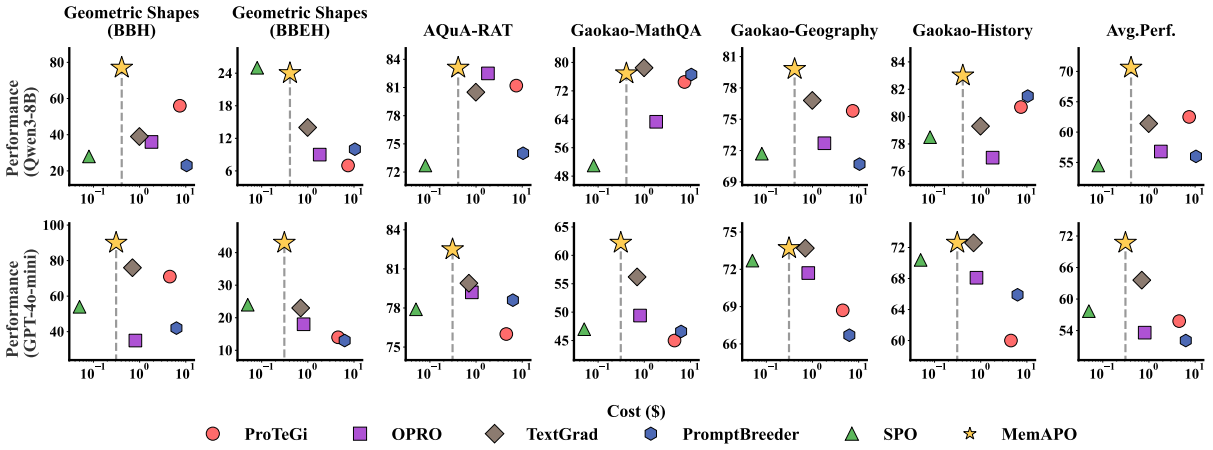

Figure 2: Comparison of performance and optimization costs across six automatic prompt optimization methods, six tasks, and two backbones. MemAPO achieves the best average performance across all datasets on both backbones, while notably reducing cost by around ***57.2%*** compared to the strong baseline TextGrad (Yuksekgonul et al., 2025).

this process, recent works have shifted towards utilizing LLMs as optimizers, which can be broadly categorized into three paradigms. (1) Text Gradient-Based Optimization (Pryzant et al., 2023; Yuksekgonul et al., 2025), which utilizes natural language feedback as optimization signals; (2) Trajectory-Based Optimization (Yang et al., 2023; Tang et al., 2025), which leverages historical prompt-score pairs to guide iterative improvement; and (3) Evolutionary-Based Optimization (Guo et al., 2023; Fernando et al., 2023), which employs mutation and selection operators on a prompt population. Despite their effectiveness, these approaches generally depend on external, more capable LLM (e.g., GPT-5) serving as the optimizer for a weaker target model (e.g., GPT-4o-mini), which limits their scalability and practical deployment.

2.2 Self-Evolving Agents

The inherent dependency on external supervision raises a fundamental question: Can a model optimize itself without external guidance? Early works (Madaan et al., 2023; Lu et al., 2023) mainly focus on self-correction and iterative refinement. Drawing inspiration from human cognitive processes, where past experiences are distilled to inform current decisions, recent works (Wei et al., 2025) have incorporated memory-augmented mechanisms to store and retrieve historical insights. From the perspective of memory abstraction, existing self-learning methods can be broadly divided into two categories. The first group (Shinn et al., 2023; Zhang et al., 2023) focuses on episodic memory, where concrete interaction trajectories are stored and reused. In contrast, the second group abstracts historical interactions into higher-level guidelines, as exemplified by (Zhao et al., 2024; Fu et al., 2024). Recently, Training-Free GRPO (Cai et al., 2025) and ACE (Zhang et al., 2025) demonstrate that LLM behavior can be self-improved by updating contexts or experiential priors. However, they learn a general playbook or a shared experience library, without the capability to explicitly model prompt optimization as query-dependent prompt composition. Building on this insight, MemAPO targets this gap by maintaining dual memories of reusable correct templates and error patterns, and retrieving them according to each query to construct adaptive prompts for heterogeneous tasks.

3 Methodology

3.1 Problem Definition

Consider a downstream task $\mathcal{D} = \{(q_i, a_i)\}_{i=1}^{n}$ consisting of n question-answer pairs, where q_i denotes the input query and a_i is the optional ground-truth answer. Let $\mathcal{M}_t$ denote the target LLM, and let $\phi(\cdot,\cdot)$ be an evaluation function that measures the quality of a generated response which can be instantiated as an exact-match evaluator for closed tasks with verifiable answers, LLM-as-Judge scorer (Gunjal et al., 2025) for open-ended tasks.

The objective of prompt optimization is to learn a prompt p that maximizes the expected task performance of the target model over the data distribution. Formally, the optimal prompt p^* can be defined as

$$p^* = \arg\max_{p} \mathbb{E}_{(q,a)\sim\mathcal{D}}\Big[\phi\big(\mathcal{M}_t(q;p), a\big)\Big], \quad (1)$$

where $\mathcal{M}_t(q;p)$ denotes the response generated by the target model conditioned on the input query

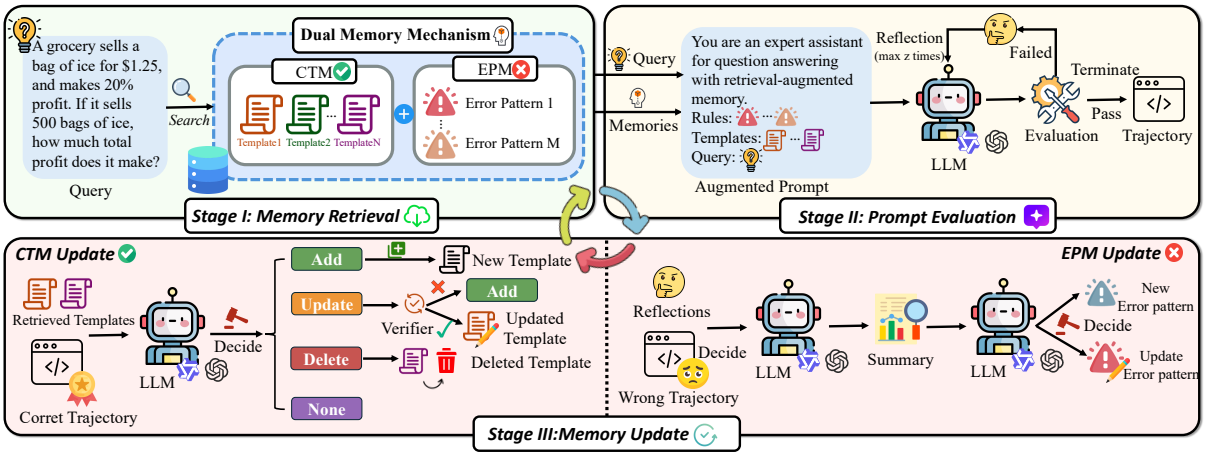

Figure 3: Overall illustration of MemAPO. (I) **Memory Retrieval**: it first retrieves relevant strategies and failure patterns according to the given query. (II) **Prompt Evaluation**: Then, the augmented prompt is constructed and iteratively evaluated. (III) **Memory Update**: Newly acquired experiences are consolidated into memory.

q and prompt p. Unlike conventional automatic prompt optimization approaches that search for a single static prompt, we model the prompt as a composition of two components $p = [p_{\text{Inst}}, p_{\text{Mem}}(q)]$. The first component p_{Inst} represents the invariant instruction that specifies general task guidelines such as reasoning style or output format. The second component $p_{\text{Mem}}(q)$ is a query-dependent memory augmentation retrieved from historical experience, which provides task-relevant strategies or constraints for the specific query q.

Under this formulation, prompt optimization can be viewed as learning how to accumulate, organize, and retrieve reusable prompting knowledge such that the memory-augmented prompt p improves the model's performance across heterogeneous queries.

3.2 MemAPO Overview

The overview architecture of MemAPO is illustrated in Figure 3. Specifically, MemAPO maintains a **Dual-Memory Mechanism** (Section 3.3) that organizes past iteration experiences into two complementary forms of knowledge: (1) *Correct templates* for transferable prompting strategies; and (2) *Error patterns* for preventing repeated mistakes. Based on this memory structure, MemAPO contains three stages:

(1) **Memory Retrieval** (Section 3.4). Given an input query, the framework retrieves relevant strategy templates and global error patterns from memory to construct a memory-augmented prompt that constrains the model's reasoning behavior.

(2) **Prompt Evaluation** (Section 3.5). The target LLM generates responses conditioned on the retrieved memory, while a self-reflection mechanism iteratively evaluates intermediate outputs and produces corrective feedback to improve reasoning.

(3) **Memory Update** (Section 3.6). Newly acquired experiences are consolidated into memory by either creating/updating templates or summarizing error patterns, enabling the memory repository to evolve over time.

During inference, MemAPO directly retrieves relevant memory and performs a single-pass response generation without invoking the reflection loop, allowing efficient deployment while benefiting from accumulated prompting knowledge.

3.3 Dual Memory Mechanism

To enable continual improvement through experience accumulation, MemAPO maintains a structured memory that abstracts reusable knowledge from historical interactions. Specifically, MemAPO organizes historical experiences into two complementary memory components: **Correct-Template Memory (CTM)**, which captures reusable successful strategies, and **Error-Pattern Memory (EPM)**, which summarizes recurring failure modes. Together, these two components provide both *positive guidance and negative constraints during prompt construction*.

Correct-Template Memory (CTM). Inspired by schema theory in cognitive science, humans interpret new situations by activating previously learned knowledge structures rather than reasoning from scratch (Rumelhart, 2017). Similarly, CTM abstracts successful interaction histories into structured templates that can be reused across semantically similar queries. Formally, the template repos-

itory is defined as $\mathcal{E}_{\text{CTM}} = \{T_1, T_2, \dots\}$. Each template is represented as $T = \{I, S, C\}$. Here, I denotes the **index**, which specifies the applicability conditions of the template and serves as a retrieval key. S denotes the **strategy**, which distills transferable reasoning procedures extracted from successful trials. By separating strategies from specific inputs, CTM enables knowledge reuse across heterogeneous queries. Finally, C denotes the set of supporting **cases**, which store verifiable exemplars that ground the abstract strategy and facilitate imitation during prompting.

Error-Pattern Memory (EPM). While CTM captures successful experiences, Error-Pattern Memory records common failure patterns observed during previous trials. Instead of storing raw incorrect responses, EPM abstracts them into high-level reflective rules that describe typical reasoning mistakes and corresponding prevention principles. Formally, the error-pattern repository is defined as $\mathcal{E}_{\text{EPM}} = \{R_1, R_2, \dots\}$. where each R represents a generalized description of a recurrent failure mode derived from reflection on erroneous outputs.

3.4 Memory Retrieval

Given a query q, the memory retrieval stage selects relevant experiences from the dual-memory repository to construct a memory-augmented prompt. Formally, the retrieval process can be expressed as

$$\mathcal{T}_q^{(k)}, \mathcal{R}_q = \mathcal{F}_{\text{Retrieval}}\left(\mathcal{E}_{\text{CTM}}, \mathcal{E}_{\text{EPM}} \,|\, q\right), \quad (2)$$

where $\mathcal{T}_q^{(k)}$ denotes the set of retrieved strategy templates from CTM and $\mathcal{R}_q$ represents the retrieved error patterns from EPM.

CTM Retrieval. To identify relevant prompting strategies, we retrieve templates from the Correct-Template Memory $\mathcal{E}_{\text{CTM}}$ according to semantic similarity between the query and template indices. Specifically, let $\mathbf{e}_q$ denote the embedding of the query and $\mathbf{e}_I$ denote the embedding of a template index I. The similarity score is computed as $f_{\text{sim}}(q, T) = \cos(\mathbf{e}_q, \mathbf{e}_I)$. The retrieved template set is defined as:

$$\mathcal{T}_q^{(k)} = \text{TopK}_{T \in \mathcal{E}_{\text{CTM}}}\, f_{\text{sim}}(q, T), \quad (3)$$

where $\mathcal{T}_q^{(k)} = \{T_q^{(1)}, \dots, T_q^{(k)}\}$ contains the top-k templates with the highest similarity scores. To avoid retrieving irrelevant strategies, we further filter templates whose similarity scores fall below a predefined threshold θ_{corr}.

EPM Retrieval. Unlike CTM, Error-Pattern Memory encodes global behavioral constraints that apply broadly across queries. Therefore, all stored error patterns are retrieved and incorporated into the prompt context $\mathcal{R}_q = \mathcal{E}_{\text{EPM}}$. These patterns provide negative guidance during reasoning by discouraging the model from reproducing previously observed failure modes.

3.5 Prompt Evaluation

Given the retrieved strategy templates $\mathcal{T}_q^{(k)}$ and error patterns $\mathcal{R}_q$, MemAPO constructs a memory-augmented prompt to guide the reasoning process of the target model. The augmented prompt is defined as $p_{\text{aug}} = [p_{\text{Inst}}, \mathcal{R}_q, \mathcal{T}_q^{(k)}]$ where p_{Inst} denotes the task instruction, while $\mathcal{R}_q$ and $\mathcal{T}_q^{(k)}$ provide negative constraints and positive reasoning strategies retrieved from memory.

To acquire informative experiences for future memory updates, MemAPO employs a self-reflection mechanism (Shinn et al., 2023) that iteratively evaluates intermediate reasoning attempts and produces corrective feedback. Let $\mathcal{L}$ denote the set of accumulated reflections, initialized as $\mathcal{L} = \{\emptyset\}$. Conditioned on the query, augmented prompt, and current reflections, the target model generates an initial attempt

$$\hat{a}_0 = \mathcal{M}_t(q, p_{\text{aug}}, \mathcal{L}; p_{\text{ans}}^{\text{meta}}), \quad (4)$$

where $p_{\text{ans}}^{\text{meta}}$ is a meta-prompt that instructs the model to incorporate retrieved memories and reflections during reasoning.

The generated answer is then evaluated using the task evaluation function. If the attempt fails (incorrect or judged as low-quality), the model is prompted to reflect on the reasoning process and produce a corrective insight

$$\text{ref} = \mathcal{M}_t(q, p_{\text{aug}}, \mathcal{L}, \hat{a}_0; p_{\text{ref}}^{\text{meta}}), \quad (5)$$

where $p_{\text{ref}}^{\text{meta}}$ denotes the meta-prompt for reflection. The newly generated reflection is incorporated into the reflection set via $\mathcal{L} \leftarrow \mathcal{L} \cup \{\text{ref}\}$. The model then retries answer generation using the updated reflection context according to Eq. 4. This self-reflection loop continues until either a correct answer is produced or the maximum number of retries is reached.

3.6 Memory Update

After prompt generation and evaluation, MemAPO updates its memory repository based on the outcome of the reasoning attempt.

CTM Update. The Correct-Template Memory evolves by incorporating successful reasoning experiences into reusable templates. If no template is retrieved for the current query, the model constructs a new template by summarizing the successful interaction. Concretely, the target model generates a new template index and strategy using a meta-prompt $p_{\text{corr}}^{\text{meta}}$:

$$(I_{\text{new}}, S_{\text{new}}) = \mathcal{M}_t(q, \hat{a}, \mathcal{L}; p_{\text{corr}}^{\text{meta}}), \quad (6)$$

where $\hat{a}$ denotes the final correct response and $\mathcal{L}$ contains the accumulated reflections. The resulting template $T_{\text{new}} = \{I_{\text{new}}, S_{\text{new}}, C_{\text{new}}\}$ is added to the CTM repository $\mathcal{E}_{\text{CTM}}$, where $C_{\text{new}} = \{(q, \hat{a})\}$ stores the corresponding interaction case.

When relevant templates are retrieved, the new successful interaction $(q, \hat{a})$ is appended to their case sets to enrich supporting examples. To maintain template quality, the retrieved templates $\mathcal{T}_q^{(k)}$, together with the interaction $(q, \hat{a})$ and reflections $\mathcal{L}$, are provided to the model via a meta-prompt $p_{\text{act}}^{\text{meta}}$ that determines how the templates should evolve. MemAPO selects one of the following operations: (1) ADD: create a new template when the current interaction represents a novel reasoning pattern. (2) UPDATE: refine an existing template by incorporating complementary strategy information. (3) DELETE: remove templates that are contradicted by newly observed evidence. (4) NONE: retain the existing template without modification. For the UPDATE operation, we perform a verification step: a subset of stored cases from the original template is sampled, and the updated strategy is accepted only if it successfully solves all sampled cases. Otherwise, the update is rejected, and a new template is created instead. Further, to prevent unbounded growth of the template repository, MemAPO periodically performs template consolidation when the number of templates exceeds a predefined capacity. Similar templates are merged to maintain a compact and diverse set of strategies.

EPM Update. While CTM records successful experiences, Error-Pattern Memory captures recurring failure modes. After a failed attempt, the accumulated reflections $\mathcal{L}$ are summarized into a generalized error pattern

$$R_{\text{sum}} = \mathcal{M}_t(q, a, \mathcal{L}; p_{\text{sum}}^{\text{meta}}), \quad (7)$$

where $p_{\text{sum}}^{\text{meta}}$ is a meta-prompt that abstracts the reflection trajectory into a concise rule describing the failure cause. To maintain a compact error memory, we identify similar error patterns in the repository according to semantic similarity

$$\mathcal{R} = \{R_i \mid f_{\text{sim}}(R_i, R_{\text{sum}}) > \theta_{\text{error}}\}, \quad (8)$$

where θ_{error} denotes a similarity threshold. If similar patterns exist, the model decides whether to refine them by incorporating the new rule; otherwise, a new error pattern is added to the repository.

4 Experiment

4.1 Experimental Setup

Datasets. We evaluate MemAPO on both closed tasks and open-ended tasks. For closed tasks, we utilize a collection of datasets spanning logical reasoning, mathematical calculation, and knowledge-intensive tasks. For logical reasoning, we adopt two established datasets, **Big-Bench Hard (BBH)** (Suzgun et al., 2023) and **Big-Bench Extra Hard (BBEH)** (Kazemi et al., 2025), and focus on the Geometric Shapes task. We use multiple subtasks from **AGIEval** (Zhong et al., 2024), including AQuA-RAT and Gaokao-Math for mathematical calculation, as well as Gaokao-Geography and Gaokao-History for knowledge-intensive reasoning. Following (Yuksekgonul et al., 2025), we sampled portions from original tasks as test sets for each sub-task. For open-ended tasks, we select *writing*, *roleplay* and *humanities* from MT-Bench (Zheng et al., 2023b) following (Xiang et al., 2025). Detailed construction procedures are provided in the Appendix A.1.

Baselines. We compare MemAPO with representative APO methods: **ProTeGi** (Pryzant et al., 2023), **OPRO** (Yang et al., 2023), **TextGrad** (Yuksekgonul et al., 2025), **PromptBreeder** (Fernando et al., 2023), **SPO** (Xiang et al., 2025). In addition, we also consider conventional prompting methods, comprising **IO** (direct inference), **CoT** (chain-of-thought) (Wei et al., 2022), **Step-Back** (Zheng et al., 2023a), and **RaR** (rephrase and respond) (Deng et al., 2023).

Implementation Details & Metrics. We conduct experiments with GPT-4o-mini and Qwen3-8B (Yang et al., 2025), respectively, using the same model as both the task executor and the prompt optimizer. We report the performance using accuracy for closed tasks, following (Suzgun et al., 2023; Kazemi et al., 2025; Zhong et al., 2024). For MT-Bench, we use GPT-4o to judge outputs and report

Table 1: Main comparison results across three domains. Tasks from similar domains are merged into a unified training set, and the optimized prompt is evaluated separately on each task. All methods use the same model for both prompt optimization and evaluation, and experiments are conducted on Qwen3-8B and GPT-4o-mini, respectively. GeoShape refers to the "Geometric Shapes" task, Avg. Perf and Avg. Cost refers to the average performance and the average optimization cost, respectively.

Backbone	Method \ Dataset	Logical Reasoning		Mathematical Calculation		Knowledge Intensive		Avg. Perf. ↑	Avg. Cost($) ↓
		GeoShape (BBH)	GeoShape (BBEH)	AQuA-RAT	Gaokao MathQA	Gaokao Geography	Gaokao History		
Qwen3-8B	IO	23.0	6.0	79.9	68.5	72.7	80.0	55.0	-
	CoT (Wei et al., 2022)	64.0	6.0	82.5	65.7	76.8	78.5	62.3	-
	Step-Back (Zheng et al., 2023a)	58.0	8.0	81.8	64.5	70.7	76.3	59.9	-
	RaR (Deng et al., 2023)	53.0	3.0	80.5	70.1	78.8	77.8	60.5	-
	ProTeGi (Pryzant et al., 2023)	56.0	7.0	81.2	74.5	75.8	80.7	62.5	7.44
	OPRO (Yang et al., 2023)	36.0	9.0	82.5	63.3	72.7	77.0	56.8	1.81
	TextGrad (Yuksekgonul et al., 2025)	39.0	14.0	80.5	**78.5**	76.8	79.3	61.4	0.99
	PromptBreeder (Fernando et al., 2023)	23.0	10.0	74.0	76.6	70.7	81.5	56.0	10.44
	SPO (Xiang et al., 2025)	28.0	**25.0**	72.7	51.0	71.7	78.5	54.5	**0.08**
	MemAPO	**77.0**	24.0	**83.1**	76.9	**79.8**	**83.0**	**70.6**	0.41
GPT-4o-mini	IO	35.0	13.0	61.7	39.4	71.7	71.9	48.8	-
	CoT (Wei et al., 2022)	47.0	17.0	69.5	35.5	70.7	67.4	51.2	-
	Step-Back (Zheng et al., 2023a)	61.0	25.0	59.7	41.8	62.6	60.7	51.8	-
	RaR (Deng et al., 2023)	43.0	30.0	66.9	38.6	67.7	58.5	50.8	-
	ProTeGi (Pryzant et al., 2023)	71.0	14.0	76.0	45.0	68.7	60.0	55.8	4.52
	OPRO (Yang et al., 2023)	35.0	18.0	79.2	49.4	71.7	68.1	53.6	0.81
	TextGrad (Yuksekgonul et al., 2025)	76.0	23.0	79.9	56.2	**73.7**	**72.6**	63.6	0.70
	PromptBreeder (Fernando et al., 2023)	42.0	13.0	78.6	46.6	66.7	65.9	52.1	6.28
	SPO (Xiang et al., 2025)	54.0	24.0	77.9	47.0	72.7	70.4	57.7	**0.05**
	MemAPO	**90.0**	**43.0**	**82.5**	**62.2**	**73.7**	**72.6**	**70.7**	0.31

win rates of MemAPO compared to other methods following (Xiang et al., 2025). We also measure the optimization cost to assess efficiency. Other details are provided in the Appendix A.2.

4.2 Experimental Results and Analysis

Main Results of Closed Tasks. Table 1 reports the performance of different APO methods under heterogeneous settings, where tasks from similar domains are jointly optimized and evaluated separately. On GPT-4o-mini, our method achieves the best average performance 70.7% across all tasks, outperforming the strongest baseline TextGrad by **7.1%**. Compared with baselines that exhibit clear trade-offs across tasks, improving some datasets while degrading others, our method consistently yields stable gains across most tasks. To further demonstrate the effectiveness of MemAPO, we conduct experiments on Qwen3-8B backbone and observe similar trends. Specifically, MemAPO achieves 70.6% average accuracy, exhibiting the best performance on most datasets. These results indicate that our MemAPO generalizes well across both proprietary and open-source models. For mixed domain tasks evaluation and learned experiences, please refer to the Appendix A.6 and A.7.

Main Results of Open-ended Tasks. We conduct win rates comparison between different LLMs and methods across *writing*, *roleplay*, and *humanities*. As shown in Figure 4, MemAPO achieves substantially higher win rates across nearly all settings,

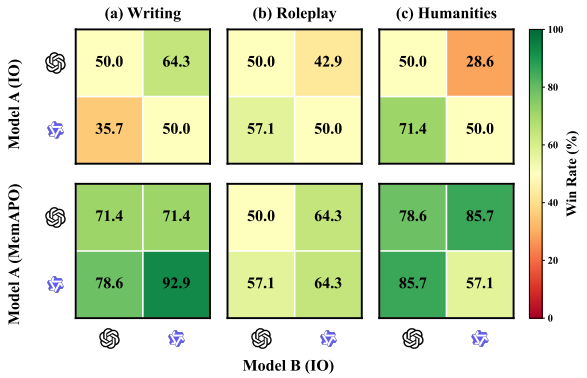

Figure 4: Win rates comparison on open-ended tasks.

demonstrating that the dual-memory mechanism effectively generalizes to open-ended generation where no ground-truth answer exists.

Cost Analysis. Besides the performance, we present a comparison of optimization costs and performance between MemAPO and other methods in Figure 2. Whether using Qwen3-8B or GPT-4o-mini as the target LLM, our method achieves superior performance while ranking second in optimization efficiency across all methods. Compared to the strong baseline TextGrad, it reduces training costs by 58.6% and 55.7%, respectively. We also provide detailed statistics of prefill/decode tokens in Appendix A.6. Although APO baselines can amortize prefill cost via KV-cache reuse of optimized prompt at inference time, MemAPO achieves substantially lower decoding cost and latency despite higher prefill tokens.

Table 2: Ablation study on the effect of dual memory components in MemAPO using GPT-4o-mini.

Correct Template Memory	Error Pattern Memory	AQuA-RAT	Gaokao MathQA
✗	✗	61.7	39.4
✗	✓	77.9	60.2
✓	✗	80.5	61.0
✓	✓	**82.5**	**62.2**

Table 3: Ablation study on the effect of self-evolving paradigm. We compare performance across different LLMs for prompt optimization.

Method	AQUA-RAT		Gaokao MathQA	
	GPT-4o-mini	GPT-5-Chat	GPT-4o-mini	GPT-5-Chat
ProTeGi	76.0	82.5	45.0	50.6
OPRO	79.2	79.2	49.4	47.0
TextGrad	79.9	81.8	56.2	60.2
PromptBreeder	78.6	80.5	46.6	36.3
SPO	77.9	79.2	47.0	63.3
MemAPO	**82.5**	**83.8**	**62.2**	**64.5**

4.3 Ablation Study

The Effect of Dual Memory Components. To verify the contribution of the two key memory components, CTM and EPM, we conduct an ablation study on the mathematical domain using GPT-4o-mini, as shown in Table 2. Both CTM and EPM individually bring substantial performance gains on AQuA-RAT and Gaokao MathQA, and combining them leads to further improvements of 2.0 and 1.2 points, respectively. These results indicate that reusable strategies and failure patterns provide complementary benefits.

The Effect of Self-Evolving Paradigm. We conduct an ablation study to validate the effectiveness of the self-evolving paradigm, as shown in Table 3. The experiment results show that introducing a stronger LLM as optimizer generally leads to performance improvements, but may sometimes have the opposite effect. For MemAPO, it achieves 98.4% and 96.4% of the performance obtained with GPT-5 when using GPT-4o-mini as optimizer, validating MemAPO's self-evolving paradigm.

The Impact of Template Number. As shown in Figure 5, we analyze the impact of Top-k templates retrieved from CTM on model performance and inference token consumption. The performance curves on AQuA-RAT and Gaokao MathQA show similar trends: As the number of templates increases, the performance initially improves and then declines, peaking at the top 3 templates. A possible explanation is that a moderate number of templates offers effective reasoning guidance, whereas retrieving too many templates introduces redundant or noisy informatio, potentially leading to instability. In terms of inference token consumption, as the number of templates increases, the input context to the LLM becomes longer, resulting in a steady increase in token usage. Based on the above analysis, we select the top-3 templates.

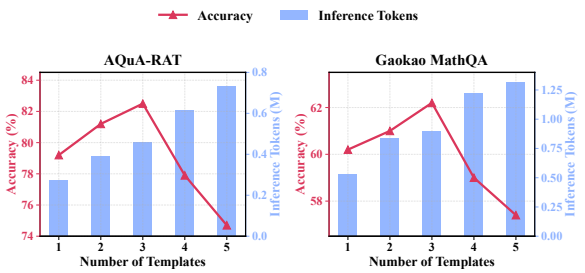

Figure 5: Ablation study on the impact of template number with GPT-4o-mini on mathematical domain.

4.4 Memory Evolution

We further investigate the scalability and stability of the memory as the volume of accumulated experiences grows. A detailed analysis in Appendix A.5 shows that the memory expands rapidly in the early stages to actively absorb useful experiences, and gradually converges to a bounded size through selective editing and capacity-triggered merging. Meanwhile, previously acquired templates retain their effectiveness throughout long-horizon inference, while newly generated templates provide complementary improvements. These findings suggest that the memory remains both scalable and stable over time, without suffering from severe knowledge obsolescence or confirmation bias.

5 Conclusion

In this paper, we revisit automatic prompt optimization from a new perspective. Instead of treating prompt optimization as a static prompt search problem, we formulate it as a continual experience accumulation process. Based on this, we propose MemAPO, a self-evolving memory that enables LLM agents to progressively accumulate and reuse prompting knowledge across heterogeneous tasks. MemAPO maintains a dual-memory mechanism that organizes successful reasoning strategies and failure signals into structured experiences, which are dynamically retrieved to guide prompt construction and refined through iterative self-reflection. Extensive experiments demonstrate that MemAPO consistently improves performance while substantially reducing optimization cost.

Limitations

While MemAPO demonstrates strong performance in automatic prompt optimization through self-evolving memory, some limitations remain. The current framework focuses on textual reasoning tasks and prompt optimization in language-based settings, without explicitly modeling multimodal information such as visual, audio, or structured environment signals. Extending the memory architecture to support multimodal experiences may further enhance the generality of the framework in broader agentic environments.

References

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, and 1 others. 2025. Training-free group relative policy optimization. *arXiv preprint arXiv:2510.08191*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. 2023. Rephrase and respond: Let large language models ask better questions for themselves. *arXiv preprint arXiv:2311.04205*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, and 1 others. 2024. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 1107–1128.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37:119919–119948.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, and 1 others. 2026. Rethinking memory mechanisms of foundation agents in the second half. *arXiv preprint arXiv:2602.06052*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Yuanzhu Peter Chen, and 1 others. 2025. Big-bench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26473–26501.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 3045–3059.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2024. Gpt understands, too. *AI Open*, 5:208–215.

Jianqiao Lu, Wanjun Zhong, Wenyong Huang, Yufei Wang, Qi Zhu, Fei Mi, Baojun Wang, Weichao Wang, Xingshan Zeng, Lifeng Shang, and 1 others. 2023. Self: Self-evolution with language feedback. *arXiv preprint arXiv:2310.00533*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 7957–7968.

David E Rumelhart. 2017. Schemata: The building blocks of cognition. In *Theoretical issues in reading comprehension*, pages 33–58. Routledge.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and 1 others. 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.

Xinyu Tang, Xiaolei Wang, Wayne Xin Zhao, Siyuan Lu, Yaliang Li, and Ji-Rong Wen. 2025. Unleashing the potential of large language models as prompt optimizers: Analogical analysis with gradient-based model optimizers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25264–25272.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, and 1 others. 2025. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*.

Jinyu Xiang, Jiayi Zhang, Zhaoyang Yu, Fengwei Teng, Jinhao Tu, Xinbing Liang, Sirui Hong, Chenglin Wu, and Yuyu Luo. 2025. Self-supervised prompt optimization. *arXiv preprint arXiv:2502.06855*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. 2023. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36:78227–78239.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, and 1 others. 2025. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023a. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023b. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2299–2314.

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. In *The eleventh international conference on learning representations*.

A Appendix

A.1 Tasks and Data Details

In this section, we present the details of the experimental dataset sizes and data splits in Table 4. Table 5 presents the details of the dataset licenses and sources. All datasets are released under the MIT License or the Apache License 2.0 and are open for academic research. All our usage strictly complies with the original intended purposes of the dataset, and no personal privacy information is included by human careful check.

Table 4: Dataset sizes and data splits.

Dataset	Train & Dev	Test
BBH-Geometric Shapes	150	100
BBEH-Geometric Shapes	100	100
AGIEval-AQuA-RAT	100	154
AGIEval-Gaokao MathQA	100	251
AGIEval-Gaokao Geography	100	99
AGIEval-Gaokao History	100	135
MT-Bench	9	71

BBH-Geometric Shapes is a sub-task from the BIG-Bench Hard dataset (Suzgun et al., 2023). This task focuses on geometric reasoning, requiring the model to determine the geometric shape that would be generated based on the given SVG path element containing multiple commands. In the experiments, we follow the dataset split used by (Yuksekgonul et al., 2025).

BBEH-Geometric Shapes is a sub-task from the BIG-Bench Extra Hard dataset (Kazemi et al., 2025) that builds upon BBH by replacing each of the 23 tasks from BBH with a novel counterpart. BBEH-Geometric Shapes is much more difficult than the BBH-Geometric Shapes because each set of given commands can draw multiple shapes and may involve many useless commands. Following the (Yuksekgonul et al., 2025), we randomly split (seed=42) the data into training, validation, and test sets with a 50:50:100 ratio.

AGIEval-AQuA-RAT is a subset of the AGIEval benchmark (Zhong et al., 2024), focusing on evaluating the mathematical problem-solving ability using questions from the GRE and GMAT exams. In the experiments, 100 samples are randomly selected (seed=42) for the training and development set, with the remaining samples used for testing. AGIEval-Gaokao MathQA, AGIEval-Gaokao Geography, and AGIEval-Gaokao History adopt the same data split to facilitate the heterogeneous datasets' construction. For brevity, we omit repeating this split strategy when introducing these tasks later.

AGIEval-Gaokao MathQA is a subset of the AGIEval (Zhong et al., 2024) that measures mathematical reasoning performance on Chinese college-entrance exam questions.

AGIEval-Gaokao Geography. This AGIEval subset (Zhong et al., 2024) targets geography-oriented question answering and is composed of problems collected from the Chinese Gaokao Geography exam.

AGIEval-Gaokao History. Within the AGIEval (Zhong et al., 2024), this subset focuses on historical understanding by evaluating responses to questions drawn from the Chinese Gaokao History exam.

MT-Bench (Zheng et al., 2023b) is a multi-turn benchmark comprising 80 high-quality open-ended questions across eight categories (e.g., writing, reasoning, math, coding, STEM, humanities), designed to evaluate the conversational and instruction-following abilities of chat assistants through multi-turn dialogues. In the experiments, we select *writing*, *roleplay*, *humanities*.

A.2 Baseline Details

In this section, we report the configurations of the baselines used for comparison with MemAPO in the experiments. Except for the analysis described in section 4.3, all LLM components serving different roles in the methods are instantiated using the same underlying model.

ProTeGi performs prompt optimization for 6 iterations with a minibatch size of 64 (Pryzant et al., 2023). At each iteration, the prompt is evaluated on sampled training examples to identify errors, which are summarized into textual gradients describing how the prompt should be improved. The prompt is then edited in the opposite semantic direction of these gradients to generate new candidate prompts. A beam search with bandit-based selection retains the top 4 prompts for the next iteration.

OPRO treats the large language model as an optimizer that iteratively generates candidate prompts based on a meta-prompt containing previously generated prompts and their performance scores (Yang et al., 2023). The iterative process runs for 20 rounds, with 8 prompts generated in each round.

Table 5: Licenses and sources of the experimental datasets.

Dataset	License	Source
BBH-Geometric Shapes	MIT License	https://github.com/suzgunmirac/BIG-Bench-Hard/blob/main/bbh/geometric_shapes.json
BBEH-Geometric Shapes	Apache License 2.0	https://github.com/google-deepmind/bbeh/tree/main/bbeh/benchmark_tasks/bbeh_geometric_shapes
AGIEval-AQuA-RAT	MIT License	https://github.com/ruixiangcui/AGIEval/blob/main/data/v1/aqua-rat.jsonl
AGIEval-Gaokao MathQA	MIT License	https://github.com/ruixiangcui/AGIEval/blob/main/data/v1/gaokao-mathqa.jsonl
AGIEval-Gaokao Geography	MIT License	https://github.com/ruixiangcui/AGIEval/blob/main/data/v1/gaokao-geography.jsonl
AGIEval-Gaokao History	MIT License	https://github.com/ruixiangcui/AGIEval/blob/main/data/v1/gaokao-history.jsonl
MT-Bench	MIT License	https://github.com/lm-sys/FastChat/blob/main/fastchat/llm_judge/data/mt_bench/question.jsonl

TextGrad conducts prompt optimization across 3 epochs with 4 steps per epoch, employing stochastic gradient descent with a batch size of 3 (Yuksekgonul et al., 2025). At each step, gradient signals are derived by back-propagating the feedback produced by the optimizer, enabling iterative refinement of the system prompt. A validation-based rollback mechanism is employed such that any update that reduces validation performance is discarded, reverting the prompt to its previous state.

PromptBreeder optimizes prompts using an evolutionary algorithm that maintains a population of task-prompts and mutation-prompts (Fernando et al., 2023). The evolution is initialized with 5 mutation prompts and 5 thinking styles, and runs for 20 generations. In each generation, the LLM generates mutated prompt variants, evaluates their performance (fitness) on training examples, and iteratively selects and evolves better prompts.

SPO performs self-supervised prompt optimization without relying on external ground-truth labels (Xiang et al., 2025). The optimization runs for 10 iterations, each iteration randomly selects 3 questions without answers from the training and validation datasets.

A.3 Implementation Details

The effectiveness of MemAPO is validated on both a closed-source commercial model (GPT-4o-mini) and an open-source model (Qwen3-8B). For the dual-memory mechanism, we implement it on top of vector databases as the underlying storage layer. Embeddings are generated with Qwen3-Embedding-8B, and semantic similarity is utilized as the retrieval criterion. In Stage I, for CTM retrieval, we retrieve the top-3 templates during both training and inference, with similarity thresholds θ_{corr} set to 0.3 and 0.1, respectively; for EPM retrieval, we retrieve all recorded error patterns, as they represent the full spectrum of error types observed in previous attempts. In Stage II, for the self-reflection mechanism, the maximum number of retries is set to 3. In Stage III, for the CTM update, we introduce an additional verification step: each updated template's strategy must succeed on all 3 cases randomly sampled from the corresponding original correct template, otherwise the system will fall back into the template creation process. Moreover, the maximum number of templates in the system is capped at 30. Once this limit is exceeded, the template merge operation will be triggered to reduce template counts; for EPM update, the similarity threshold θ_{error} is set to 0.7. In the experiments we report, we conducted a single run.

Initial Prompt. For the initial prompt, we use the original ones from (Suzgun et al., 2023) for logical reasoning tasks, "Let's solve the problem" from (Tang et al., 2025) for mathematical calculation and knowledge-intensive tasks.

A.4 Experimental Environments Details

For all models used in this paper, we accessed them through their official APIs. The parameter sizes of closed-source models, such as GPT-4o-mini and GPT-5-Chat, are not publicly available. For the open-source setting, we employ an 8B large language model and an 8B embedding model, respectively. The total API compute budget was under one thousand US dollars. All experiments were conducted on a Linux-based system. The memory model implementation and reproduction are built based on the open-sourced vector stores ChromaDB.

A.5 Memory Evolution

To verify that the dual memory mechanism genuinely evolves, we analyze its dynamic changes during optimization on the mathematical calculation domain using GPT-4o-mini.

Memory Growth and Convergence. Table 6 reports the memory size and retrieval statistics at different optimization stages. Both CTM and EPM grow rapidly in the early stages, indicating active accumulation of reusable strategies and failure patterns. After 75% of progress, CTM growth satu-

rates (28 templates), while the average number of retrieved templates per query continues to increase (from 0.94 to 2.06). This suggests that the memory becomes more reusable over time, rather than growing unboundedly.

Table 6: Memory growth and retrieval dynamics during optimization progress.

Optimization Progress	CTM Size	EPM Size	Avg. Retrieved Templates
25%	9	7	0.94
50%	21	10	1.60
75%	28	14	1.79
100%	28	14	2.06

Active Memory Maintenance. During optimization, the CTM is actively maintained through 33 ADD operations, 26 UPDATE operations, and 16 DELETE operations, while 144 cases require no modification because existing templates already generalize well. Among the 26 UPDATE attempts, 20 pass validation, whereas 6 attempts are rejected. In addition, the merge operation is triggered 6 times, with 5 merge groups accepted and 1 rejected, ultimately reducing the CTM size from 36 to 28 without degrading performance.

Long-Horizon Template Utility. To assess whether older templates remain useful or are superseded by newer ones, we partition all templates by creation time into three equal groups (oldest, middle, and most recent) and measure their correct use rate during optimization. Templates from all three periods contribute meaningfully, achieving correct use rates of 66.6%, 69.1%, and 76.2%, respectively. These results suggest that early-stage templates retain long-term utility, while newer templates provide complementary improvements, indicating that the memory does not suffer from severe confirmation bias or knowledge obsolescence.

A.6 More Experiments

Computation Consumption. The specific values, including averaged API calls, token consumption, and optimization cost, are shown in table 7.

Generalization Analysis. We have discussed the results under an intra-domain setting where tasks from the same domain are jointly optimized. A more challenging and practical scenario arises when mixing tasks from different domains. We construct a heterogeneous dataset by selecting Gaokao Math QA and Gaokao History from the mathematical calculating and knowledge-intensive domains,

Table 7: Comparison of averaged API calls, token consumption, and optimization costs across different methods and backbones on Qwen3-8B and GPT-4o-mini.

Method	Avg. Call↓	Avg. Token (M)↓	Avg. Cost($)↓
Qwen3-8B			
ProTeGi	15682	17.14	7.44
OPRO	4849	3.97	1.81
TextGrad	2016	2.86	0.99
PromptBreeder	27660	32.76	10.44
SPO	**75**	**0.37**	**0.08**
MemAPO	469	1.60	0.41
GPT-4o-mini			
ProTeGi	15451	13.27	4.52
OPRO	2915	2.28	0.81
TextGrad	2059	1.92	0.70
PromptBreeder	27664	22.65	6.28
SPO	**75**	**0.30**	**0.05**
MemAPO	490	1.39	0.31

Table 8: Cross-domain (Gaokao MathQA and Gaokao History) experiment using GPT-4o-mini.

Method	Gaokao MathQA	Gaokao History
IO	39.4	71.9
CoT	35.5	67.4
StepBack	41.8	60.7
Rephrase	38.6	58.5
ProTeGi	45.0	53.3
OPRO	56.2	67.4
TextGrad	59.0	69.6
PromptBreeder	37.5	71.1
SPO	35.9	71.9
MemAPO	**61.8**	**72.6**

respectively. As shown in Table 8, most baselines suffer from severe performance degradation, exemplified by SPO, whose performance drops 11.1% on Gaokao MathQA. This indicates that a specific optimized prompt struggles to reconcile the conflicting reasoning demands of different domains. In contrast, MemAPO maintains improvements on two tasks, achieving the best performance, which demonstrates the generalization ability. For more experiments, see Table 9 and 10.

KV Cache Analysis. Representative automatic prompt optimization methods can reuse the kv cache of the optimized prompt, while MemAPO requires recalculating the entire attention matrices for the query and the augmented prompt. Therefore, we conduct token consumption comparison on AQuA-RAT based on Qwen3-8B. For static-prompt baselines, we cache the optimized prompt and count only query-side prefill tokens. For MemAPO, we count the full query-dependent augmented prompt. As shown in Figure 11, MemAPO has higher prefill cost but much lower decoding cost and the best performance under this strict setting, making it applicable to real scenarios.

Table 9: Cross-domain (BBH-GeoShape and AQuA-RAT) experiment from Math to History using GPT-4o-mini.

Method	GeoShape	AQuA-RAT
IO	35.0	61.7
CoT	47.0	69.5
Step-Back	61.0	59.7
RaR	43.0	66.9
ProTeGi	66.0	68.2
OPRO	35.0	76.6
TextGrad	79.0	79.2
PromptBreeder	45.0	73.4
SPO	51.0	74.7
MemAPO	**88.0**	**81.2**

Table 10: Cross-domain (BBH-GeoShape and Gaokao History) experiment using GPT-4o-mini.

Method	GeoShape	Gaokao History
IO	35.0	71.9
CoT	47.0	67.4
Step-Back	61.0	60.7
RaR	67.7	58.5
ProTeGi	68.0	60.0
OPRO	39.0	67.4
TextGrad	74.0	71.1
PromptBreeder	42.0	66.7
SPO	53.0	69.6
MemAPO	**89.0**	**72.6**

Table 11: Token consumption comparison between static optimized prompt methods and MemAPO.

Method	Prefill	Decode	Performance
ProTeGi	18,107	101,750	81.2
OPRO	18,107	104,696	82.5
TextGrad	18,107	190,568	80.5
PromptBreeder	18,107	105,003	74.0
SPO	18,107	137,984	72.7
MemAPO	341,258	48,144	83.1

Example Error Patterns

rules:
1. Always ensure that calculations align with the context and constraints of the problem, particularly when selecting from finite answer choices.
2. Always validate that derived values and relationships adhere to original constraints and check for consistency.
...

Figure 6: Example error patterns.

A.7 Examples of Learned Experiences

Figure 6 shows typical error patterns that should be avoided in math domain. Figure 7 presents an example of a learned template for vector-related mathematical problems. The template summarizes the applicable problem conditions, key solving strategy, and representative solved cases for reference.

A.8 Meta Prompt

In this section, we present the meta prompts used in section 3 in Figure 8, 9, 10, 11, 12, 13, 14. These meta prompts serve as the core instructions that drive each stage of the MemAPO pipeline, guiding the target LLM to perform memory-augmente answering, self-reflection, template creation and management, and error pattern summarization. We provide them in full to ensure reproducibility.

A.9 Ethical Statement

A.9.1 Scientific Artifacts

All data sources and models underlying the experiment are used with explicit references and official links. Detailed information can be found in Appendix A.1, A.2. The experiments primarily target English and Chinese language questions across three domains: logical reasoning, mathematical problem solving, and knowledge-intensive question answering. The codes and experiment results will be released publicly with clear documentation, permitting use for research purposes. We manually collected the dataset according to its original requirements and, through manual review, excluded any data related to personal privacy.

A.9.2 Potential Risks

MemAPO is designed for efficient and accurate prompt optimization, it not only improves task performance but also reduces optimization costs. However, potential risks may arise from unintended misuse, such as over-interpreting experiment results as indicators of real-world deployment readiness or being maliciously exploited to generate harmful or misleading content.

A.10 Use of AI Assistants

LLMs are used in this work strictly as auxiliary tools for limited language polishing of the manuscript, including improving fluency and presentation. All technical content, experimental analysis, and scientific claims are authored and finalized by the authors.

Example Template

Index: This template should be applied when solving problems involving vector magnitudes, angles between vectors, and dot products with given vector relationships.
Strategy: 1. Identify and list the given information about the vectors. 2. Calculate the magnitude of known vectors if necessary. 3. Calculate the dot product of the vectors, or use the cosine formula to relate the dot product and angle between vectors: $\cos(\theta) = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|}$. 4. Ensure all mathematical relationships and conditions are satisfied.
Case 1
Question: Given vector $\vec{a} = (2, 1)$, $\vec{a} \cdot \vec{b} = 10$, and $|\vec{a} + \vec{b}| = 5\sqrt{2}$, find $|\vec{b}|$. Options: (A) $\sqrt{5}$ (B) $\sqrt{10}$ (C) 5 (D) 25.
Answer: To solve the problem, we need to utilize the given information about the vectors $\vec{a}$ and $\vec{b}$.
1. **Given Information**: $\vec{a} = (2, 1)$, $\vec{a} \cdot \vec{b} = 10$, and $|\vec{a} + \vec{b}| = 5\sqrt{2}$.
2. **Finding** $|\vec{a}|$: First, calculate the magnitude of vector $\vec{a}$:

$$|\vec{a}| = \sqrt{2^2 + 1^2} = \sqrt{4 + 1} = \sqrt{5}.$$

3. **Expanding the Magnitude of** $|\vec{a} + \vec{b}|$: We know the formula for the magnitude of the sum of two vectors:

$$|\vec{a} + \vec{b}|^2 = |\vec{a}|^2 + |\vec{b}|^2 + 2(\vec{a} \cdot \vec{b}).$$

Substituting the known values:

$$(5\sqrt{2})^2 = |\vec{a}|^2 + |\vec{b}|^2 + 2(\vec{a} \cdot \vec{b}).$$

This gives:

$$50 = 5 + |\vec{b}|^2 + 2 \cdot 10.$$
$$|\vec{b}|^2 = 25, \qquad |\vec{b}| = 5.$$

4. **Conclusion**: We have calculated $|\vec{b}|$ correctly while ensuring all mathematical conditions were satisfied, particularly the relationship between the magnitudes and the dot product. Thus, the correct answer is (C) 5.
Case 2: ...
Case 3: ...

Figure 7: An example retrieved template.

Answering Query Meta Prompt

You are an expert assistant for question answering with retrieval-augmented memory.
Your job: answer the user's question by leveraging retrieved TEMPLATES as guidance and strictly following RULES derived from historical errors.

{init_instruction}

RULES
The following rules are summarized from historical errors. You MUST follow them strictly:
{rules}

<TEMPLATES>
Below are retrieved templates. Use their strategies as guidance. Each template includes one verified good case.
{templates}
</TEMPLATES>

<REFLECTIONS>
You have attempted this question before and failed. Learn from your mistakes and do NOT repeat them.
{reflections}
</REFLECTIONS>

<QUESTION>
{question}
</QUESTION>

<OUTPUT_FORMAT>
{output_format}
</OUTPUT_FORMAT>
Before providing the final answer, outline your corresponding problem-solving steps, and avoid the mistakes mentioned in the rules. Finally, present your final answer according to the required output format.

Figure 8: Meta prompt for answering query.

Self Reflection Meta Prompt

You are a precise self-reflection assistant.
Your job: analyze why the previous answer was wrong and extract a concise, actionable lesson.

Your previous answer to the following question is likely WRONG. Re-examine your reasoning and find the mistake.

<QUESTION>
{question}
</QUESTION>

<RULES>
The following rules are summarized from historical error
{rules}
</RULES>

<TEMPLATES>
Below are retrieved templates.
{templates}
</TEMPLATES>

<REFLECTIONs>
{reflections}
</REFLECTIONs>

<YOUR_ANSWER>
{wrong_pred}
</YOUR_ANSWER>

Instructions:
1. Re-read the question carefully and check whether your answer actually addresses all constraints and conditions.
2. Trace through your reasoning step by step — identify any logical gaps, unjustified assumptions, or calculation errors.
3. Consider alternative interpretations or approaches you may have overlooked.
4. If prior reflections exist, do NOT repeat the same analysis — dig deeper or try a completely different angle.

You MUST respond with a JSON object in exactly this format and nothing else:

```
{
    "analysis": "your detailed step-by-step analysis of what went wrong",
    "reflection": "one-sentence actionable lesson to avoid this mistake next time"
}
```

Figure 9: Meta prompt for self-reflection.

Template Creation Meta Prompt

You are an expert at abstracting reusable problem-solving templates.
Your job: given a question and its correct answer, extract a generalizable template that can guide solving similar problems in the future.

A model answered the following question correctly. Abstract this success into a reusable template.

<QUESTION>
{question}
</QUESTION>

<REFLECTIONs>
{reflections}
</REFLECTIONs>

<CORRECT_ANSWER>
{correct_pred}
</CORRECT_ANSWER>
{reflections_block}

Instructions:
1. Analyze the question type, scenario characteristics, and what makes this kind of problem recognizable.
2. Abstract the general solution procedure based on the successful reasoning trajectory. Describe the key reasoning or analysis steps needed to solve this type of problem. Each step should represent a high-level reasoning operation and can be directly reusable for a new problem that matches the same scenario. Keep the steps minimal, non-redundant, and sufficient for a single-pass solution attempt.
3. If prior failed attempts and reflections are provided, incorporate the lessons learned as pitfalls to avoid in the strategy.
4. Produce:
   - "when_to_use": a concise description of WHEN this template should be applied (what kind of question/scenario triggers it)
   - "strategy": the abstracted step-by-step reasoning procedure for this type of problem, including pitfalls to avoid if reflections are available

You MUST respond with a JSON object in exactly this format and nothing else:
{
   "when_to_use": "one-sentence description of the applicable scenario",
   "strategy": "concise general reasoning strategy for this type of problem"
}

Figure 10: Meta prompt for template creation.

Template Update Meta Prompt

You are an expert template manager for a retrieval-augmented problem-solving system.
Your job: given a new successfully solved case and the recalled templates, decide the best action to keep the template library accurate, non-redundant, and maximally useful.

A model just answered a question correctly. Review the recalled templates and decide what actions to take.

<RECALLED_TEMPLATES>
{recalled_templates}
</RECALLED_TEMPLATES>

<NEW_GOOD_CASE>
question: {question}
correct_answer: {correct_pred}
</NEW_GOOD_CASE>
{reflections_block}
You must decide an action for EACH recalled template, and optionally add new templates. Rules:
- Each recalled template must appear EXACTLY ONCE in the actions list.
- Each action targets ONE template.

Available actions:
1. **none**: The recalled template already covers this case well (semantically equivalent). Keep it unchanged. Specify the template_id.
2. **update**: The recalled template is relevant but its when_to_use or strategy can be enriched / made more comprehensive with information from the new case. Specify the template_id and provide updated fields.
   - "when_to_use": a concise description of WHEN this template should be applied (what kind of question/scenario triggers it). Set to null to keep unchanged.
   - "strategy": the abstracted step-by-step reasoning procedure for this type of problem, including pitfalls to avoid if reflections are available. Set to null to keep unchanged.
3. **delete**: The recalled template conflicts with the new case (e.g. wrong strategy, contradictory advice) or is fully superseded. Specify the template_id.
4. **add**: The new case represents a genuinely new problem type not covered by ANY recalled template. Create a new template. (Use sparingly — only when none of the recalled templates can be updated to cover this case.)
   - "when_to_use": a concise description of WHEN this template should be applied (what kind of question/scenario triggers it).
   - "strategy": the abstracted step-by-step reasoning procedure for this type of problem, including pitfalls to avoid if reflections are available.

IMPORTANT:
- Only use template_id values that appear in RECALLED_TEMPLATES above. Do NOT invent template_ids.
- Every recalled template_id must appear exactly once across all actions.

You MUST respond with a JSON object containing an "actions" list:
```
{
   "actions": [
      {"action": "none", "template_id": "..."},
      {"action": "update", "template_id": "...", "when_to_use": "... or null", "strategy": "... or null"},
      {"action": "delete", "template_id": "..."},
      {"action": "add", "when_to_use": "...", "strategy": "..."}
   ]
}
```


Figure 11: Meta prompt for template update.

Template Merge Meta Prompt

You are an expert template librarian for a retrieval-augmented problem-solving system.
Your job: given a full list of templates, identify groups of templates that are semantically similar or overlapping and can be merged into a single, more general template without losing coverage.

The template library has grown too large ({total}) templates, limit is {limit}). Identify groups of templates that can be merged to reduce the total count.

<ALL_TEMPLATES>
{all_templates}
</ALL_TEMPLATES>

Instructions:
1. Read ALL templates carefully. Identify groups where the when_to_use scenarios overlap significantly or the strategies are highly similar / complementary.
2. Only merge templates that are truly related — do NOT force-merge unrelated templates just to reduce count.
3. For each merge group, produce a single merged template that covers all the scenarios and combines the best parts of each strategy.
4. Templates NOT included in any merge group will be kept as-is.
5. Try to reduce the total template count to at most {target} through merging.

You MUST respond with a JSON object in exactly this format and nothing else:

```
{
  "merge_groups": [
    {
      "template_ids": ["1", "3"],
      "reason": "brief explanation of why these templates should be merged",
      "merged_when_to_use": "combined scenario description covering all merged templates",
      "merged_strategy": "combined strategy incorporating the best of each template"
    }
  ]
}
```

If no templates can be reasonably merged, return: {"merge_groups": []}

Figure 12: Meta prompt for template merge.

Error Summarization Meta Prompt

You are an expert error-pattern analyst.
Your job: given a question, its correct answer, and multiple failed attempts with reflections, synthesize ONE concise, generalizable error-pattern description that can prevent similar mistakes in the future.

A model attempted the following question multiple times and FAILED every time. Analyze all attempts holistically and extract the root-cause error pattern.

<QUESTION>
{question}
</QUESTION>

<CORRECT_ANSWER>
{correct_pred}
</CORRECT_ANSWER>

<FAILED_ATTEMPTS>
{failed_attempts}
</FAILED_ATTEMPTS>

Instructions:
1. Compare all failed attempts — identify the COMMON root cause, not just surface-level symptoms.
2. Consider whether the errors stem from misunderstanding the question, flawed reasoning, calculation mistakes, or missing domain knowledge and so on.
3. Abstract the lesson into a generalizable rule that applies beyond this specific question.

You MUST respond with a JSON object in exactly this format and nothing else:

```
{
   "root_cause": "brief description of the common root cause across all attempts",
   "reflection": "one-sentence generalizable rule to prevent this category of error in the future"
}
```

Figure 13: Meta prompt for error summarization.

Error Pattern Update Meta Prompt

You are an expert error-pattern analyst.
Your job: refine an existing error-pattern description by incorporating new evidence from bad cases. The updated pattern must be concise, generalizable, and actionable — it will be used as a RULE to prevent similar mistakes in the future.

An existing error pattern needs to be updated because a new bad case has been added to its cluster.

<CURRENT_PATTERN>
{current_pattern}
</CURRENT_PATTERN>

<HISTORICAL_BAD_CASES>
The following are existing bad cases already in this error pattern cluster.
{historical_bad_cases}
</HISTORICAL_BAD_CASES>

<NEW_BAD_CASE>
This is the new bad case that triggered the update.
question: {new_question}
correct_answer: {new_ground_truth}
wrong_answer: {new_wrong_pred}
reflection: {new_reflection}
</FAILED_ATTEMPTS>

Instructions:
1. Read the current pattern and historical bad cases to understand the existing error pattern.
2. Analyze the NEW bad case — determine whether it introduces a genuinely new dimension to the pattern.
3. If the new bad case is very similar to the historical ones and the current pattern already covers it well, you may keep the current pattern UNCHANGED.
4. Otherwise, produce a refined, generalizable one-sentence pattern description that:
   - Covers BOTH the new bad case and the historical ones
   - Is more precise or more general than the current pattern if the new evidence warrants it
   - Is actionable — it should clearly state what to do or avoid

You MUST respond with a JSON object in exactly this format and nothing else:
{
   "analysis": "brief reasoning about whether the new bad case changes or confirms the pattern",
   "updated": "true or false (whether the pattern needs to be updated)"
   "pattern": "one-sentence error-pattern description (refined if updated=true, or the original current pattern if updated=false)"
}

Figure 14: Meta prompt for error pattern update.