Beyond Meta-Reasoning: Metacognitive Consolidation for Self-Improving LLM Reasoning

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated strong reasoning capabilities, and as existing approaches for enhancing LLM reasoning continue to mature, increasing attention has shifted toward meta-reasoning as a promising direction for further improvement. However, most existing meta-reasoning methods remain episodic: they focus on executing complex meta-reasoning routines within individual instances, but ignore the accumulation of reusable meta-reasoning skills across instances, leading to recurring failure modes and repeatedly high metacognitive effort. In this paper, we introduce Metacognitive Consolidation, a novel framework in which a model consolidates metacognitive experience from past reasoning episodes into reusable knowledge that improves future meta-reasoning. We instantiate this framework by structuring instance-level problem solving into distinct roles for reasoning, monitoring, and control to generate rich, attributable meta-level traces. These traces are then consolidated through a hierarchical, multi-timescale update mechanism that gradually forms evolving meta-knowledge. Experimental results demonstrate consistent performance gains across benchmarks and backbone models, and show that performance improves as metacognitive experience accumulates over time.

1 Introduction

Large Language Models (LLMs) have demonstrated strong reasoning capabilities, and extensive work has further advanced them through reinforcement learning (OpenAI et al., 2024; Guo et al., 2025) and test-time scaling (Snell et al., 2025; Wu et al., 2025). As these traditional routes mature, attention is increasingly shifting to meta-reasoning, the ability to reason about how to reason, which offers a complementary perspective for improving the intelligence of LLMs (Wang and Zhao, 2024; Yan et al., 2025).

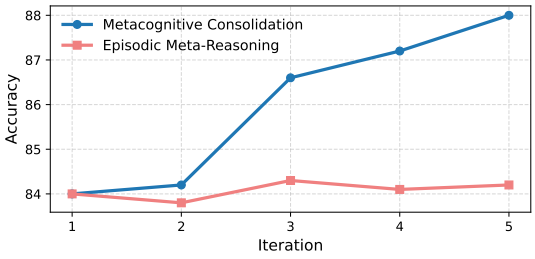

Figure 1: **Metacognitive Consolidation** goes beyond episodic meta-reasoning by accumulating experience across instances, leading to steadily improving performance as test-time experience grows (blue), whereas traditional meta-reasoning remains episodic and exhibits limited or fluctuating gains (red). (benchmark: MATH-500)

Existing research on LLM meta-reasoning largely follows two paradigms. The first is pre-task meta-planning, where the model generates guiding meta-information, such as a reasoning plan, strategy, or scaffolding, before executing the actual reasoning (Gao et al., 2024; Suzgun and Kalai, 2024; Wan et al., 2025). The second is in-task meta-control, where the model enters reasoning directly but injects meta-level interventions during the trajectory to steer, revise, or re-plan intermediate steps (Zhang et al., 2025; Yang et al., 2025; Xu et al., 2025).

However, most existing meta-reasoning methods remain episodic. They focus solely on the execution of meta-reasoning acts within a single instance, ignoring the acquisition of meta-reasoning skills over time. As a result, similar failure modes recur on recurring problems, and the system repeatedly relies on the same heavy meta-reasoning routines (e.g., retries, verification, or intervention) to recover (Tan et al., 2025). This lack of skill carryover prevents amortizing metacognitive effort over time, leaving the model largely *cognitively static* rather than evolving into a progressively more robust reasoner (He et al., 2025).

To address this challenge, we introduce **Metacognitive Consolidation**. Beyond performing meta-reasoning, a model should consolidate meta-level experience accumulated across reasoning episodes into reusable procedural knowledge that improves future meta-reasoning. Theoretically, this echoes the cognitive account of *proceduralization* (Anderson, 1982), where skills transition from an explicit, declarative stage to a procedural stage in which the same competence can be executed more directly with practice. For LLMs, this mechanism is pivotal as it converts episodic test-time compute into persistent capability growth, enabling the model to evolve from a static solver into a self-improving reasoner.

To enable metacognitive consolidation, we require structured and attributable meta-level feedback about what failed and how it was corrected, whereas standard Chain-of-Thought (Wei et al., 2022) often entangles such signals into an opaque, monolithic stream. Accordingly, we propose **Meta-Reasoning Orchestrator** (MRO), a modular architecture in which three specialized agents collaboratively augment the reasoning process with explicit meta-level structure. The *Monitor* audits the *Reasoner*'s trajectory and the *Controller* intervenes based on this feedback, consistent with cognitive accounts of meta-reasoning as monitoring and control (Ackerman and Thompson, 2017). This factorization yields fine-grained action–critique–correction traces that support metacognitive consolidation while also improving inference-time reliability through explicit oversight and targeted intervention.

To realize metacognitive consolidation, we introduce **MetaCognitive Accumulator** (MCA), a hierarchical memory module that retains and updates meta-level experience across reasoning instances at multiple temporal frequencies. The MCA aggregates *instance-level reflections* into *batch-level micro-lessons*, and further integrates them into *long-term meta-knowledge* that evolves more slowly over time. This hierarchical accumulation enables continual adaptation while balancing retention and forgetting, reflecting the role of bounded, multi-timescale memory in modern reasoning systems (Behrouz et al., 2025a,b).

Together, MRO and MCA form an inner–outer loop architecture, named as MC^2 (MetaCognitive Consolidation), where MRO performs instance-level meta-reasoning and MCA accumulates and updates meta-knowledge across instances. To demonstrate the effectiveness of our method, we evaluate it across multiple backbone models and reasoning benchmarks, showing consistent improvements over strong baselines. Moreover, as shown in Figure 1, these improvements grow with experience, indicating that the model progressively accumulates and leverages metacognitive knowledge over time.

Our contributions are threefold:

- **New Framework**: We introduce Metacognitive Consolidation, a new perspective that extends meta-reasoning beyond episodic, instance-level control to the accumulation of reusable metacognitive skills.
- **Framework Instantiation**: We propose MC^2, an inner–outer loop framework consisting of a Meta-Reasoning Orchestrator (MRO) and a MetaCognitive Accumulator (MCA), enabling structured meta-reasoning within instances and hierarchical consolidation across instances.
- **Empirical Performance**: Through extensive experiments across multiple models and reasoning benchmarks, we show that metacognitive consolidation yields consistent gains that grow with experience.

2 Method

We consider a reasoning stream $\mathcal{D} = \{(x_t, y_t^\star)\}_{t=1}^{T}$, where x_t is an input instance and $y_t^\star$ is the gold answer. Let $\mathrm{LLM}_\theta(\cdot)$ denote a frozen backbone model. Given a prompt, the model generates an answer $\hat{y}_t$ together with a reasoning trajectory τ_t (e.g., a chain-of-thought).

Episodic reasoning. In the standard (episodic) setting, each instance is solved independently with a fixed inference prompt P:

$$(\hat{y}_t, \tau_t) = \mathrm{LLM}_\theta(x_t \mid P). \tag{1}$$

Test-time evolvement. In contrast, under *test-time evolvement*, the inference policy is allowed to *update online* as the system processes more instances. We model this by maintaining a time-indexed policy P_t that is updated using experience from previous reasoning trajectories:

$$P_{t+1} = \mathrm{update}(P_t, \tau_t), \tag{2}$$

and then used to solve the next instance:

$$(\hat{y}_{t+1}, \tau_{t+1}) = \mathrm{LLM}_\theta(x_{t+1} \mid P_{t+1}). \tag{3}$$

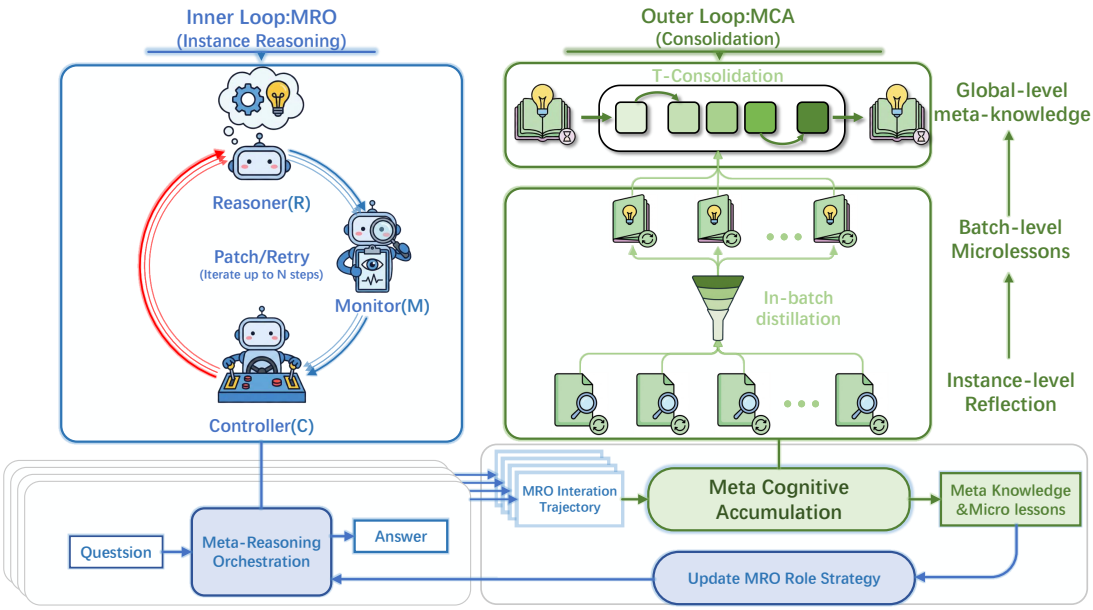

Figure 2: Overview of **Metacognitive Consolidation (MC^2)**. The inner loop (MRO) produces structured action–critique–correction traces via a Reasoner–Monitor–Controller decomposition, while the outer loop (MCA) consolidates these traces across instances into evolving meta-knowledge and updates role-specific policies for subsequent inference.

Overall framework. As shown in Figure 2, MC^2 consists of two coupled modules. **Meta-Reasoning Orchestrator (MRO)** is the inner loop that performs instance-level meta-reasoning by coordinating a *Reasoner*, *Monitor*, and *Controller*, producing both the final answer and an explicit action–critique–correction trace. **MetaCognitive Accumulator (MCA)** is the outer loop that consolidates these traces across instances into meta-knowledge at multiple temporal frequencies and uses it to form stronger role policies over time.

2.1 Meta-Reasoning Orchestrator

To consolidate the metacognitive experience, the system needs structured and attributable signals about *what failed* and *what fixed it*. In contrast, standard chain-of-thought reasoning (Wei et al., 2022) primarily exposes the solution trajectory itself, while many self-revision pipelines (e.g., self-refinement or verification) often entangle critique and correction into a single opaque stream (Madaan et al., 2023), making it harder to extract reusable meta-level signals.

Motivated by the cognitive science view of meta-reasoning as *monitoring and control of thinking* (Ackerman and Thompson, 2017), we decompose instance-level inference into three specialized agents Reasoner/Monitor/Controller with role-specific policy prompts $P_t \triangleq (P_{R,t}, P_{M,t}, P_{C,t})$. Within the MRO, the Reasoner proposes a solution trajectory, the Monitor audits it, and the Controller decides whether to accept, patch, or request a revision. This interaction is implemented as an iterative loop: the Controller's feedback is passed back to the Reasoner to guide the next attempt, and the loop repeats until a termination condition is met or a maximum of N iterations is reached.

Reasoner. At inner iteration k, the Reasoner generates a candidate answer and trajectory conditioned on the input and the Controller's previous feedback u_t^{k-1}:

$$(\hat{y}_t^k, \tau_t^k) = \mathcal{R}\left(x_t, u_t^{k-1} \mid P_{R,t}\right), \tag{4}$$

where $u_t^{-1} = \emptyset$ for initialization. Intuitively, u_t^{k-1} can encode targeted constraints such as "re-check step j" or "restart with an alternative approach."

Monitor. Given the Reasoner's proposal, the Monitor audits the trajectory and produces a diagnostic report:

$$g_t^k = \mathcal{M}\left(x_t, \hat{y}_t^k, \tau_t^k \mid P_{M,t}\right), \tag{5}$$

where g_t^k typically includes (i) a verdict, (ii) an error localization signal (e.g., the first suspicious step), and (iii) a concise critique or evidence.

Controller. The Controller decides whether to accept, patch, or restart based on the candidate and the Monitor's report:

$$(a_t^k, \tilde{y}_t^k, u_t^k) = \mathcal{C}\Big(x_t, \hat{y}_t^k, \tau_t^k, g_t^k \mid P_{C,t}\Big), \quad (6)$$

where the action $a_t^k \in \{\textsc{Accept}, \textsc{Patch}, \textsc{Restart}\}$, $\tilde{y}_t^k$ is an optional patched answer (only used when $a_t^k = \textsc{Patch}$), and u_t^k is feedback passed to the next Reasoner call (only used when $a_t^k = \textsc{Restart}$).

The MRO loop terminates when either (i) $a_t^k \in \{\textsc{Accept}, \textsc{Patch}\}$ or (ii) $k = N-1$ (iteration budget exhausted). We define the resulting structured trace as:

$$\mathcal{T}_t = \Big\{\big(\hat{y}_t^k, \tau_t^k, g_t^k, a_t^k\big)\Big\}_{k=0}^{K_t-1}, \quad K_t \le N. \quad (7)$$

This factorization yields fine-grained action–critique–correction records that are directly usable for consolidation, while also improving inference-time robustness via explicit oversight and targeted intervention.

2.2 MetaCognitive Accumulator

MCA consolidates the meta-level experience produced by MRO into reusable knowledge that improves future meta-reasoning. A key challenge is that raw traces $\{\mathcal{T}_t\}$ are heterogeneous and noisy across instances, and cannot be directly "stored" as meta-knowledge without redundancy and drift. We therefore adopt a *hierarchical, multi-frequency* consolidation scheme with three levels: *instance-level reflections* $\rightarrow$ *batch-level micro-lessons* $\rightarrow$ *global-level meta-knowledge*.

Since Reasoner, Monitor, and Controller serve distinct roles, MCA processes their experiences *separately* and maintains three role-specific consolidation streams. As the consolidation pipeline is identical across roles, we omit the $\mathcal{R}/\mathcal{M}/\mathcal{C}$ subscripts in what follows and describe the generic MCA procedure for a single role.

Instance-level reflection. For each instance, MCA compresses the role-specific trace into a lightweight, attributable reflection:

$$r_t = \mathrm{Ref}(x_t, \mathcal{T}_t). \quad (8)$$

Here Ref is a deterministic extractor that produces a short structured record from $\mathcal{T}_t$, including (i) iteration count K_t, (ii) outcome-aware quality tags (e.g., `task_quality`, `task_outcome`, and role-specific `quality` levels), and (iii) role-wise episode summaries of answers/diagnoses/actions across iterations. These reflections serve as the inputs to subsequent reflection distillation.

Batch-level micro-lesson. To facilitate experience accumulation across multiple instances, we partition the reasoning stream into batches $\mathcal{B}_b$ of size m. Within each batch $\mathcal{B}_b$, reflections are abundant and vary in usefulness. We therefore perform *in-batch distillation* to select informative reflections and summarize them into fewer micro-lessons. We use a simple heuristic that leverages the MRO outcomes without requiring gold labels:

$$\mathcal{S}_b^+ = \{t \in \mathcal{B}_b : K_t = 1 \wedge a_t^0 = \textsc{Accept}\},$$
$$\mathcal{S}_b^- = \{t \in \mathcal{B}_b : K_t = N \wedge a_t^{N-1} = \textsc{Restart}\}. \quad (9)$$

Here $\mathcal{S}_b^+$ contains "best" instances that succeed immediately, while $\mathcal{S}_b^-$ contains "worst" instances that exhaust the inner-loop budget and still require restarting. We then distill reflections from both sets into a concise batch-level micro-lesson:

$$\ell_b = \mathrm{Distill}\big(\{r_t\} | t \in \mathcal{S}_b^+ \cup \mathcal{S}_b^-\big). \quad (10)$$

Including both successful and failed cases provides complementary signals: successes suggest reusable tactics, while failures highlight negative patterns to avoid.

Global-level meta-knowledge. Micro-lessons should inform a long-term meta-knowledge state, but naively accumulating all past lessons can cause unbounded growth and outdated rules. Inspired by recent findings on bounded test-time memory and multi-timescale learning (Behrouz et al., 2025b,a), we adopt *temporal consolidation*. Let $\mathrm{Win}_w(\cdot)$ keep the most recent w batches of micro-lessons. We update global-level meta-knowledge as:

$$K_b = \mathrm{Consol}\Big(K_{b-1}, \mathrm{Win}_w(\{\ell_j\}_{j=1}^b)\Big). \quad (11)$$

This bounded update balances retention and forgetting: recent, repeatedly supported lessons are preserved, while stale or low-utility rules naturally decay.

Updating MRO policies. After updating the meta-knowledge at the end of batch $\mathcal{B}_b$, we apply it to guide inference in the next batch $\mathcal{B}_{b+1}$. Concretely, for each instance $x_i \in \mathcal{B}_b$, the

MRO uses an updated policy prompt that integrates the current global meta-knowledge together with instance-relevant micro-lessons, thereby conditioning instance-level meta-reasoning on accumulated experience from previous batches. The retrieval step is:

$$\tilde{\mathcal{L}}_t = \text{Retrieve}\Big(x_i,\ \mathcal{L}_b^{(w)} \mid k\Big)\,. \tag{12}$$

where $\mathcal{L}_b^{(w)}$ is windowed batches of micro-lessons. The instance-conditioned prompt compilation is:

$$P_{t+1} \leftarrow \text{update}\Big(P_t, K_b, \tilde{\mathcal{L}}_t, x_i\Big)\,. \tag{13}$$

The overall alogrithm is summarized in the Appendix A. All the policy prompts in MRO and procedure prompts in MCA are summarized in the Appendix H.

3 Experimental Setup

Datasets. We evaluate MC^2 on a suite of mathematical and symbolic reasoning benchmarks. Our main experiments use four datasets: **GSM8K** (Cobbe et al., 2021), a grade-school math word-problem benchmark emphasizing multi-step arithmetic; **MATH-500** (Hendrycks et al., 2021), a challenging subset of the MATH dataset covering diverse competition-style problems; **TheoremQA** (Chen et al., 2023), a theorem-centric QA benchmark that tests formal and conceptual mathematical reasoning; and **Game-of-24** (Yao et al., 2023), a symbolic search task where models must construct valid arithmetic expressions to reach a target value.

Baselines. We compare our framework against both standard reasoning methods and representative meta-reasoning and memory-based approaches. For **standard reasoning**, we include Chain-of-Thought (**CoT**) (Wei et al., 2022), CoT with Self-Consistency (**CoT-SC**) (Wang et al., 2023), and Tree-of-Thought (**ToT**) (Yao et al., 2023). For **meta-reasoning and memory-based** baselines, we include **Meta-Prompting** (Suzgun and Kalai, 2024), **Meta-Reasoner** (Sui et al., 2025), Test-time Prompt Intervention (**TTPI**) (Yang et al., 2025), and **Buffer-of-Thought** (Yang et al., 2024). To ensure a fair comparison, we *re-implement* all baselines under a unified evaluation protocol. Full implementation and hyperparameter details are reported in Appendix B.

Backbone Models. We evaluate all methods on four backbone models to test robustness across model families and access regimes. Specifically, we consider two close-sourced LLMs, **GPT-4o-mini** and **Gemini-2.0-flash**, and two open-weight instruction-tuned models, **Llama-3-8B-Instruct** and **Qwen3-8B**. We keep decoding settings consistent across methods whenever applicable. Exact prompting templates and decoding hyperparameters are summarized in Appendix C. x

Evaluation Metrics. We use accuracy as the primary evaluation metric across all benchmarks, and additionally report the number of decoding tokens as a measure of efficiency. To reduce randomness and improve statistical reliability, we run each method three times with different random seeds and report the mean performance along with 95% confidence intervals.

4 Experimental Results

In this section, we present comprehensive experimental results and analyses to assess the effectiveness of MC^2. We organize our study around the following research questions (**RQs**):

- **RQ1**: How does our framework perform compared to standard reasoning and meta-reasoning baselines across datasets and backbone models?
- **RQ2**: Does the framework exhibit *improved reasoning performance over time*?
- **RQ3**: How do different architectural components (MRO, MCA, and their variants) contribute to the overall performance?
- **RQ4**: What does the learned meta-knowledge capture, and how does it evolve with time?

4.1 Main Results

To answer **RQ1**, we compare MC^2 with standard reasoning baselines (CoT, CoT-SC, ToT) and representative meta-reasoning/memory methods on GSM8K, MATH-500, TheoremQA, and Game-of-24 across four backbone models. Table 1 reports accuracy (with 95% confidence intervals) and token usage. We summarize four key findings.

	GSM8K		MATH-500		TheoremQA		Game-of-24	
	Accuracy	#Tokens	Accuracy	#Tokens	Accuracy	#Tokens	Accuracy	#Tokens
				GPT-4o-mini				
CoT	91.38 (±0.60)	358 (±2)	72.67 (±2.01)	625 (±14)	44.04 (±1.70)	716 (±1)	7.67 (±1.43)	553 (±15)
CoT-SC	<u>94.44</u> (±2.26)	1805 (±18)	<u>78.93</u> (±1.25)	3200 (±37)	<u>46.92</u> (±0.95)	3643 (±4)	27.00 (±0.00)	2461 (±28)
ToT	94.41 (±0.44)	1016 (±78)	77.60 (±3.44)	3473 (±579)	45.33 (±0.95)	5774 (±200)	33.33 (±5.17)	3801 (±494)
Meta Reasoning	93.96 (±0.57)	361 (±7)	76.40 (±1.49)	657 (±16)	43.62 (±1.35)	752 (±8)	20.33 (±7.99)	964 (±151)
Meta Prompt	93.40 (±1.23)	1042 (±25)	75.73 (±3.31)	1727 (±149)	44.62 (±1.64)	2265 (±70)	<u>41.33</u> (±6.25)	3178 (±474)
Buffer of Thought	93.78 (±1.00)	1373 (±5)	75.33 (±1.03)	1756 (±19)	44.42 (±3.17)	1867 (±15)	19.33 (±3.79)	1539 (±8)
TTPI	92.50 (±1.47)	874 (±20)	74.33 (±3.53)	1208 (±72)	44.42 (±2.35)	1484 (±67)	28.00 (±4.30)	916 (±183)
MC^2	**96.92** (±0.11)	833 (±21)	**85.13** (±1.74)	1631 (±129)	**55.96** (±1.47)	2159 (±143)	**88.67** (±8.72)	2479 (±542)
				Gemini-2.0-flash				
CoT	95.58 (±0.72)	218 (±4)	91.53 (±0.29)	602 (±42)	59.34 (±0.89)	813 (±26)	72.00 (±7.45)	599 (±75)
CoT-SC	<u>96.03</u> (±0.48)	1100 (±5)	94.20 (±2.77)	3018 (±161)	60.12 (±1.13)	4025 (±45)	80.67 (±7.59)	3085 (±55)
ToT	95.98 (±0.37)	558 (±54)	<u>94.93</u> (±2.35)	1999 (±162)	<u>61.33</u> (±1.56)	4743 (±201)	83.33 (±1.43)	5549 (±887)
Meta Reasoning	95.83 (±0.94)	219 (±2)	91.73 (±1.15)	638 (±46)	59.54 (±4.35)	860 (±40)	81.00 (±4.30)	1682 (±547)
Meta Prompt	95.43 (±0.66)	1111 (±135)	91.07 (±1.03)	1465 (±147)	57.50 (±1.12)	1832 (±175)	<u>88.00</u> (±2.48)	1592 (±269)
Buffer of Thought	95.96 (±0.39)	1324 (±4)	92.27 (±3.31)	2192 (±64)	59.17 (±2.41)	2532 (±10)	78.00 (±7.45)	2158 (±117)
MC^2	**97.17** (±0.57)	573 (±23)	**96.73** (±0.57)	1299 (±111)	**68.04** (±2.92)	1753 (±22)	**94.33** (±1.43)	1219 (±216)
				Qwen3-8B				
CoT	93.45 (±1.21)	305 (±2)	84.40 (±2.17)	932 (±37)	54.33 (±1.17)	1022 (±44)	48.00 (±4.30)	2960 (±492)
CoT-SC	94.41 (±0.29)	1537 (±8)	87.93 (±0.76)	4199 (±63)	55.25 (±0.30)	4882 (±50)	<u>68.67</u> (±1.43)	7325 (±385)
ToT	94.09 (±0.39)	877 (±52)	87.53 (±0.76)	3232 (±203)	57.29 (±1.00)	5686 (±243)	61.67 (±12.25)	6562 (±636)
Meta Reasoning	94.06 (±0.29)	306 (±6)	85.27 (±1.15)	978 (±89)	49.92 (±0.95)	888 (±36)	47.33 (±1.43)	2818 (±227)
Meta Prompt	<u>94.62</u> (±0.37)	522 (±12)	<u>91.93</u> (±1.52)	1545 (±22)	<u>61.29</u> (±0.19)	1448 (±30)	48.00 (±4.97)	7393 (±4340)
Buffer of Thought	93.68 (±0.39)	1316 (±8)	86.60 (±1.31)	1983 (±71)	54.44 (±0.68)	2171 (±22)	45.67 (±5.17)	2880 (±364)
TTPI	89.74 (±1.09)	300 (±3)	83.73 (±0.29)	888 (±140)	52.29 (±2.63)	907 (±107)	58.67 (±12.25)	6410 (±1453)
MC^2	**96.66** (±0.66)	726 (±32)	**93.07** (±1.60)	1582 (±242)	**64.88** (±0.93)	1924 (±164)	**80.67** (±3.79)	4445 (±1838)
				Llama-3.1-8B-Instruct				
CoT	85.39 (±1.44)	528 (±126)	56.27 (±1.25)	1942 (±1012)	39.50 (±1.24)	1097 (±312)	16.67 (±7.59)	14734 (±2112)
CoT-SC	87.29 (±0.95)	1936 (±204)	61.47 (±3.19)	7652 (±424)	43.08 (±2.87)	8010 (±231)	24.00 (±6.57)	30741 (±2086)
ToT	87.67 (±0.76)	1478 (±106)	60.47 (±2.35)	6450 (±1036)	43.79 (±0.89)	8296 (±684)	33.00 (±4.97)	15249 (±706)
Meta Reasoning	86.55 (±1.92)	447 (±24)	54.67 (±2.91)	2117 (±526)	48.80 (±1.56)	2052 (±111)	32.67 (±1.43)	12631 (±998)
Meta Prompt	<u>90.44</u> (±1.85)	1217 (±310)	<u>65.47</u> (±0.29)	4426 (±677)	<u>53.79</u> (±3.62)	3709 (±1054)	<u>41.00</u> (±8.96)	22895 (±5859)
Buffer of Thought	86.68 (±0.61)	1745 (±367)	55.80 (±3.58)	5564 (±472)	49.92 (±1.40)	2662 (±93)	28.67 (±3.79)	6146 (±352)
TTPI	90.17 (±0.72)	1426 (±278)	54.80 (±16.36)	4284 (±1327)	36.29 (±2.82)	3757 (±839)	33.00 (±4.30)	15106 (±3139)
MC^2	**95.55** (±1.87)	1320 (±585)	**76.80** (±8.36)	4435 (±1955)	**61.46** (±2.29)	4899 (±992)	**45.33** (±5.74)	12808 (±2935)

Table 1: Main results (with 95% confidence intervals). Best/second-best are highlighted by **bold**/<u>underline</u> in Accuracy. Tokens are rounded to integers.

Consistent improvements across datasets and backbones. MC^2 achieves the best accuracy across all backbone & benchmark combinations, demonstrating that metacognitive consolidation generalizes well across tasks and models. Representative gains include GPT-4o-mini on MATH-500 ($78.93 \rightarrow 85.13$) and TheoremQA ($46.92 \rightarrow 55.96$), as well as Llama-3.1-8B-Instruct on MATH-500 ($65.47 \rightarrow 76.80$). These results indicate that consolidating meta-reasoning experience yields robust performance improvements.

Significant gains on control-intensive tasks. Improvements on GSM8K are relatively modest due to strong baseline performance, whereas MC^2 shows substantial advantages on benchmarks that require verification and backtracking. The effect is most significant on Game-of-24, where MC^2 outperforms the strongest baseline on GPT-4o-mini ($41.33 \rightarrow 88.67$). This suggests that MC^2 is particularly effective when episodic meta-reasoning repeatedly incurs similar correction costs.

Stronger benefits for weaker backbones. MC^2 provides larger relative improvements for weaker backbones while still benefiting stronger ones. For example, although Gemini-2.0-flash already performs well on MATH-500, MC^2 still improves accuracy from 94.93% to 96.73%. This pattern indicates that structured meta-reasoning and cross-instance consolidation can compensate for limited intrinsic robustness while complementing strong base models.

Good accuracy-efficiency trade-off. While MC^2 incurs higher token usage than one-pass baselines, it is more efficient than compute-heavy test-time scaling methods. On GPT-4o-mini MATH-500, MC^2 uses 1631 tokens compared to 3200 (CoT-SC) and 3473 (ToT), while achieving higher accuracy. Compared to memory-based approaches such as Buffer-of-Thought, MC^2 often attains better accuracy with comparable or lower token usage, striking a practical balance between performance and efficiency.

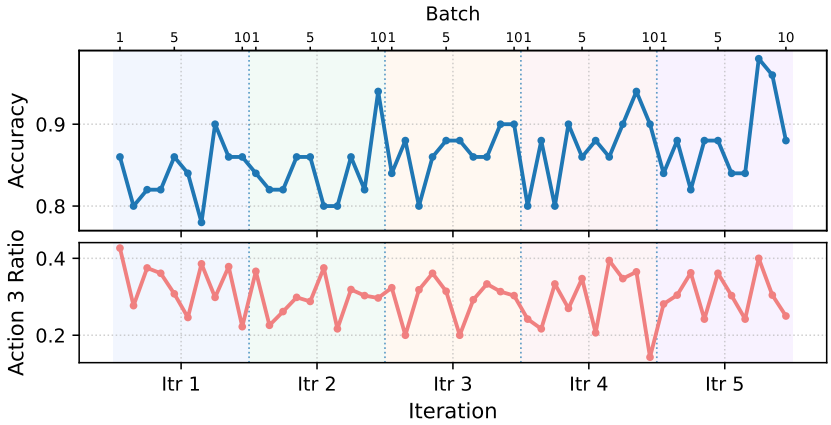

Figure 3: Top: Accuracy across batches and iterations. Bottom: Ratio of severe restart actions (Action 3) across batches and iterations.

4.2 Time Evolvement Results

To answer **RQ2**, we evaluate how performance and efficiency evolve as MC^2 processes more data and accumulates metacognitive knowledge over time. Specifically, we run MC^2 on the MATH-500 with GPT-4o-mini for multiple passes, and record accuracy and action statistics across both batches and iterations. Due to the heterogeneous difficulty of instances across batches, batch-level performance exhibits noticeable fluctuations, even though experience is continuously accumulated.

Despite batch-level variability, iteration-level trends show clear and consistent improvement. As shown in Figure 3, overall accuracy increases steadily from about 84% at iteration 1 to 88% at iteration 5, while the proportion of severe restart actions (Action 3) decreases from 33.1% to 30.8%. These results indicate that MC^2 becomes both more accurate and more stable as metacognitive knowledge accumulates. Importantly, this improvement reflects a form of *batch-level scaling*: instead of allocating more compute within a single instance, MC^2 amortizes computation across instances within a batch, enabling performance gains even without access to ground-truth supervision.

4.3 Ablation Studies

To answer **RQ3**, we conduct ablation studies to disentangle the effects of instance-level meta-reasoning (MRO) and cross-instance consolidation (MCA). We evaluate four variants: *Only Reasoner*, *R+M+C*, *R+M+C+Micro-update*, and the *Full Method*, on Math-500 and TheoremQA with two backbone models (Table 2).

Method	Math-500	TheoremQA
GPT-4o-mini		
Only Reasoner	72.67	44.04
R+ M + C	79.20	47.38
R+ M + C + Micro-update	82.80	51.75
Full Method	**85.13**	**55.96**
Qwen-3-8B		
Only Reasoner	84.40	54.33
R+ M + C	91.33	59.62
R+ M + C + Micro-update	92.60	62.21
Full Method	**93.07**	**64.88**

Table 2: Ablation study on Math-500 and TheoremQA under different backbone models.

Overall, performance improves consistently as components are added. Enabling the full MRO loop (*R+M+C*) already yields substantial gains over *Only Reasoner*, showing that structured monitoring and control directly enhance inference. Adding a simple memory mechanism (*Micro-update*) further improves accuracy, while the *Full Method* performs best, demonstrating effectiveness of MCA's hierarchical consolidation.

4.4 Case Study

To answer **RQ4**, we present a representative case on MATH-500 with GPT-4o-mini as backbone in Figure 4. In this example, the Reasoner produces a wrong answer when prompted without meta-knowledge. After incorporating the learned meta-knowledge and micro-lessons into the prompt, the Reasoner directly arrives at the correct solution without requiring additional intervention. This case demonstrates the effectiveness of meta-knowledge–guided prompt rewriting in improving reasoning accuracy. Additional examples are provided in Appendix E.

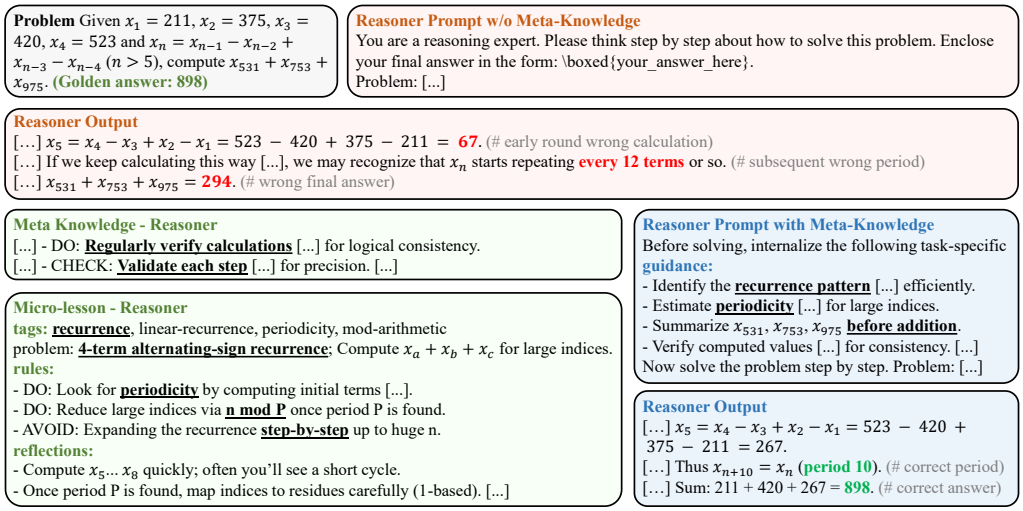

Figure 4: Case study. Red text highlights erroneous steps or final answers, while green text indicates corrected reasoning and the correct answer. Gray text denotes comments or explanations. For clarity, less relevant content is omitted and replaced with [...].

5 Related Work

LLM Meta-Reasoning Meta-reasoning refers to reasoning about how to reason, providing an additional axis for improving LLM reasoning. Most work can be grouped into two paradigms: *pre-task meta-planning* and *in-task meta-control*. Meta-Prompting (Suzgun and Kalai, 2024) and related scaffolding methods (Gao et al., 2024) structure subsequent reasoning, while ReMA (Wan et al., 2025) learns to generate meta-level guidance via multi-agent learning. Other work maintains and evolves a pool of meta-thoughts for test-time scaling and selection (Liu et al., 2025). In contrast, in-task meta-control injects meta-level interventions during reasoning. Test-time Prompt Intervention (Yang et al., 2025) modifies the reasoning process based on online signals. RLVMR (Zhang et al., 2025) encourages verification and re-planning via verifiable rewards. Hierarchical frameworks such as Thinker (Xu et al., 2025) coordinate multi-turn search, while Meta-Reasoner (Sui et al., 2025) provides dynamic inference-time guidance. Recent position paper has also highlighted the need for principled meta-reasoning from a Bayesian perspective (Yan et al., 2025). While prior methods focus on *episodic* meta-reasoning within individual instances, our work targets *Metacognitive Consolidation*, accumulating reusable meta-reasoning skills across instances as persistent meta-knowledge.

Memory for Reasoning. Another related direction improves reasoning by equipping models (or agents) with memory that reuses past reasoning artifacts. For example, ReasoningBank (Ouyang et al., 2025) and Reflexion (Shinn et al., 2023) store experience from previous rollouts (e.g., trajectories and reflections) to guide future behavior. Buffer of Thoughts (Yang et al., 2024) maintains a retrievable buffer of distilled thought templates, and ReasonFlux (Zou et al., 2025) builds hierarchical libraries of reusable reasoning patterns to reduce redundant exploration. These approaches primarily store *task-level* reasoning content (solutions, templates, or heuristics for solving), whereas we consolidate *meta-level* experience, how to monitor, critique, and control reasoning, into procedural meta-knowledge that progressively improves future meta-reasoning.

6 Conclusion

In this paper, we propose Metacognitive Consolidation, a learning framework that enables models to transform metacognitive experience from past reasoning episodes into reusable procedural knowledge for future meta-reasoning. Our framework structures instance-level reasoning into explicit roles and consolidates the resulting meta-level traces across instances via hierarchical updates. Experiments show consistent gains across benchmarks and models, with performance improving as experience accumulates. Future work will explore extensions to multi-agent settings and mechanisms for internalizing consolidated meta-knowledge more deeply into model parameters.

Limitations

We identify three key limitations of our framework.

First, our current formulation primarily consolidates *task-specific* metacognitive knowledge by repeatedly processing instances drawn from the same benchmark or task distribution. In contrast, human metacognition naturally transfers across tasks and domains. An important direction for future work is to study *task-agnostic* or cross-task metacognitive consolidation, where reusable meta-skills learned in one setting can generalize to and accelerate reasoning in new tasks.

Second, metacognitive knowledge in our framework is represented explicitly at test time, via externalized meta-knowledge and prompt-based policy updates, rather than being internalized into model parameters. Due to computational constraints, we do not explore mechanisms that allow meta-knowledge to directly influence or modify model parameters at test time. Recent advances in test-time memorization suggest that such parameter-level or hybrid forms of metacognitive consolidation may be feasible, and exploring these directions remains an important avenue for future research.

Third, enabling metacognitive consolidation incurs additional inference overhead, as the framework introduces multiple reasoning roles and periodic consolidation steps. Although our experiments demonstrate favorable performance–efficiency trade-offs relative to existing meta-reasoning methods, further optimization is needed. In particular, exploring more efficient reasoning representations, such as implicit chains of thought, may help reduce computational cost while preserving the benefits of metacognitive consolidation.

References

Rakefet Ackerman and Valerie A. Thompson. 2017. Meta-reasoning: Monitoring and control of thinking and reasoning. *Trends in Cognitive Sciences*, 21(8):607–617.

John R Anderson. 1982. Acquisition of cognitive skill. *Psychological review*, 89(4):369.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. 2025a. Nested learning: The illusion of deep learning architectures. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. 2025b. Titans: Learning to memorize at test time. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. TheoremQA: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901, Singapore. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Peizhong Gao, Ao Xie, Shaoguang Mao, Wenshan Wu, Yan Xia, Haipeng Mi, and Furu Wei. 2024. Meta reasoning for large language models. *arXiv preprint arXiv:2406.11698*.

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*.

Yancheng He, Shilong Li, Jiaheng Liu, Weixun Wang, Xingyuan Bu, Ge Zhang, Z.y. Peng, Zhaoxiang Zhang, Zhicheng Zheng, Wenbo Su, and Bo Zheng. 2025. Can large language models detect errors in long chain-of-thought reasoning? In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18468–18489, Vienna, Austria. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Qin Liu, Wenxuan Zhou, Nan Xu, James Y. Huang, Fei Wang, Sheng Zhang, Hoifung Poon, and Muhao Chen. 2025. Metascale: Test-time scaling with evolving meta-thoughts. *Preprint*, arXiv:2503.13447.

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 *Thirty-seventh Conference on Neural Information Processing Systems*.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, and 243 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2025. Reasoningbank: Scaling agent self-evolving with reasoning memory. *Preprint*, arXiv:2509.25140.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Yuan Sui, Yufei He, Tri Cao, Simeng Han, Yulin Chen, and Bryan Hooi. 2025. Meta-reasoner: Dynamic guidance for optimized inference-time reasoning in large language models. *Preprint*, arXiv:2502.19918.

Mirac Suzgun and Adam Tauman Kalai. 2024. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*.

Hexiang Tan, Fei Sun, Sha Liu, Du Su, Qi Cao, Xin Chen, Jingang Wang, Xunliang Cai, Yuanzhuo Wang, Huawei Shen, and Xueqi Cheng. 2025. Too consistent to detect: A study of self-consistent errors in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4755–4765, Suzhou, China. Association for Computational Linguistics.

Ziyu Wan, Yunxiang LI, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. 2025. ReMA: Learning to meta-think for LLMs with multi-agent reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yuqing Wang and Yun Zhao. 2024. Metacognitive prompting improves understanding in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1914–1926, Mexico City, Mexico. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2025. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations*.

Jun Xu, Xinkai Du, Yu Ao, Peilong Zhao, Yang Li, Ling Zhong, Lin Yuan, Zhongpu Bo, Xiaorui Wang, Mengshu Sun, and 1 others. 2025. Thinker: Training llms in hierarchical thinking for deep search via multi-turn interaction. *arXiv preprint arXiv:2511.07943*.

Hanqi Yan, Linhai Zhang, Jiazheng Li, Zhenyi Shen, and Yulan He. 2025. Position: LLMs need a bayesian meta-reasoning framework for more robust and generalizable reasoning. In *Forty-second International Conference on Machine Learning Position Paper Track*.

Chenxu Yang, Qingyi Si, Mz Dai, Dingyu Yao, Mingyu Zheng, Minghui Chen, Zheng Lin, and Weiping Wang. 2025. Test-time prompt intervention. *arXiv preprint arXiv:2508.02511*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin CUI. 2024. Buffer of thoughts: Thought-augmented reasoning with large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. 2025. Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. *arXiv preprint arXiv:2507.22844*.

Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. 2025. Reasonflux-PRM: Trajectory-aware PRMs for long chain-of-thought reasoning in LLMs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Appendix

A Overall Algorithm

Algorithm 1 summarizes MC^2. At a high level, we (i) retrieve top-k micro-lessons and compile instance-conditioned role prompts before each inference (Eqs. (12)–(13)), (ii) run MRO to produce per-instance traces $\mathcal{T}_t$ via iterative monitoring and control (Eqs. (4)–(7)), and (iii) consolidate traces across instances into multi-frequency memory (Eqs. (8)–(11)).

Algorithm 1 Metacognitive Consolidation (MC^2)

Require: Dataset/stream $\mathcal{D}$, batch size m, max inner iters N, window size w, retrieval top-k

Initialize base role prompts P_R, P_M, P_C and meta-knowledge states $\{K_{0,i}\}_{i\in\{R,M,C\}}$
for batch $b = 1, 2, \ldots$ **do**
  for each instance $x_t \in \mathcal{B}_b$ **do**
    for role $i \in \{R, M, C\}$ **do**
      if $b = 1$ **then**
        $\tilde{P}_{t,i} \leftarrow P_i$ ▷ cold start: no retrieval / meta-knowledge
      else
        $\mathcal{L}^{(w)}_{b-1,i} \leftarrow \mathsf{Win}_w(\{\ell_{j,i}\}_{j=1}^{b-1})$
        $\tilde{\mathcal{L}}_{t,i} \leftarrow \mathsf{Retrieve}_i(x_t; \mathcal{L}^{(w)}_{b-1,i}, k)$ ▷ top-k micro-lessons
        $\tilde{P}_{t,i} \leftarrow \mathsf{UpdPrompt}_i(P_i, K_{b-1,i}, \tilde{\mathcal{L}}_{t,i}, x_t)$ ▷ compile instance-conditioned role prompt; Eq. (13)
      end if
    end for
    Run MRO for at most N iterations to obtain $(\hat{y}_t, \mathcal{T}_t)$ using Eqs. (4)–(7) with compiled prompts $\tilde{P}_t = (\tilde{P}_{t,R}, \tilde{P}_{t,M}, \tilde{P}_{t,C})$
    for role $i \in \{R, M, C\}$ **do**
      Compute instance reflection $r_{t,i}$ via Eq. (8)
    end for
  end for
  for role $i \in \{R, M, C\}$ **do**
    Select $\mathcal{S}_b^+, \mathcal{S}_b^-$ via Eq. (9)
    Distill micro-lesson $\ell_{b,i}$ via Eq. (10)
    Update meta-knowledge $K_{b,i}$ via Eq. (11)
  end for
end for

B Baselines

This section describes our baseline re-implementations to minimize confounds between *method* differences and *implementation* differences.

We compare our framework against both standard reasoning methods and representative meta-reasoning and memory-based approaches. For **standard reasoning**, we include Chain-of-Thought (**CoT**) (Wei et al., 2022), which elicits step-by-step reasoning; CoT with Self-Consistency (**CoT-SC**) (Wang et al., 2023), which improves reliability via sampling and majority voting; and Tree-of-Thought (**ToT**) (Yao et al., 2023), which performs structured search over intermediate thoughts. For **meta-reasoning** baselines, we include **Meta-Prompting** (Suzgun and Kalai, 2024), which provides task-agnostic scaffolding to guide reasoning; **Meta-Reasoner** (Sui et al., 2025), which dynamically adjusts inference-time reasoning strategies based on the current reasoning state with a contextual multi-armed bandits; Test-time Prompt Intervention (**TTPI**) (Yang et al., 2025), which injects interventions during inference to steer the reasoning trajectory; and **Buffer-of-Thought** (Yang et al., 2024), which maintains and retrieves reusable thought templates to guide new instances. To ensure a fair comparison, we *re-implement* all baselines under a unified evaluation protocol.

Since some prior works do not release full code/prompts or rely on inaccessible closed-source evaluation settings, we follow a best-effort reproduction strategy: we adhere to each method's core inference procedure (e.g., sampling, search, intervention, retrieval) and implement end-to-end inference under a unified input–output protocol.

Unified I/O protocol and answer parsing. All baselines share the same answer extraction and normalization. Concretely, we only parse predictions from an explicit final-answer field (e.g., `Final Answer: <answer>`). If the output is invalid or unparsable, we apply the method's prescribed fallback (e.g., retry/selection rules), and otherwise mark the attempt as failure under the shared parsing rule. Unless stated otherwise, we keep the same backbone model, decoding settings, and stopping criteria across baselines to reduce confounds.

B.1 Standard reasoning baselines

CoT (Chain-of-Thought). We implement the standard "step-by-step reasoning → final answer" pipeline. The prompt instructs the model to produce intermediate reasoning and then output a single, parseable answer after a fixed marker (the final-answer field). Evaluation parses only the final-answer field and applies basic normalization to match dataset annotations.

CoT-SC (Self-Consistency). Using the same CoT prompt, we draw 5 independent samples per input and aggregate the parsed final answers. We first normalize equivalent answers, then perform majority voting. If some samples are unparsable, we deterministically select the candidate that is (i) parseable and (ii) has the highest support among parseable candidates (ties broken deterministically).

ToT (Tree-of-Thought). We reproduce ToT with a breadth-first tree search that iterates *(generate candidate thoughts → self-evaluate/rank → keep top-k to expand)* until reaching a maximum depth or a termination condition. We use: 5 candidates per node, top-2 expansion, and maximum depth 2. We replace task-specific evaluators with a generic self-evaluation prompt using the same backbone model, and record the generation/evaluation prompts and stopping rules.

B.2 Meta-reasoning and memory-augmented baselines

Meta-Prompting. We reproduce the multi-role collaboration paradigm of meta-prompting: the same backbone LLM plays a Meta Model (controller) and multiple Expert roles, distinguished only by role-specific prompt templates. Inference maintains a growing message history. At each round, the Meta Model generates the next action from the full history; if it issues an expert instruction, we extract it and call the corresponding expert in an isolated format: `Expert X: "'..."'`. We then append the expert response back to history and continue. If the Meta Model outputs the final-answer marker, we parse and return it; otherwise we follow the original procedure by appending an error message and iterating. Expert calls follow the "fresh eyes" setting: experts only see the triple-quoted instruction, not the full history. Unlike the original work, we do not implement the Python Expert / interpreter, as our benchmarks do not involve code; we therefore remove code-execution components and retain only the single-expert-per-round, task decomposition, and expert verification rules.

Meta-Reasoning. We adopt the seven reasoning strategies provided in the Meta-Reasoning work and use the same backbone model to select which strategy to apply at test time. When the selected strategy is ToT or CoT-SC, we use the same hyperparameters as specified above.

TTPI (Test-time Prompt Intervention). We reproduce TTPI's dynamic intervention procedure by segmenting generation into step-level stages. At the end of each stage, we construct multiple candidate continuations corresponding to different intervention trigger sets (e.g., favoring progression, summarization, verification, or conclusion) and select one branch as the next step, repeating until a final answer is produced. TTPI requires per-token log probabilities to compute perplexity (PPL), and the full version also uses internal-layer differences to compute the Reasoning Depth Score (RDS) for branch selection. Due to API constraints, we do not evaluate TTPI on Gemini-2.0-flash (no per-token log probabilities). On GPT-4o-mini, per-token log probabilities are available but internal-layer access is not; thus, in the *Which* module we select branches using PPL only, while keeping the rest of the intervention procedure and stopping rules unchanged.

Buffer-of-Thought (BoT). We reproduce BoT's *distill–retrieve–instantiate–update* pipeline. We first apply a *Problem Distiller* to convert each problem into a structured distilled representation (e.g., key facts, goals, constraints) for retrieval and reasoning. We then retrieve the most relevant thought template from the Meta-buffer by embedding similarity between the distilled representation and template descriptions; if similarity is below a threshold, we treat it as a new task and fall back to a generic coarse-grained template. Next, an *Instantiation* prompt combines the retrieved template with the distilled representation to guide structured reasoning and produce the final answer. Finally, a *Buffer-manager* extracts a reusable thought template from the solved trajectory and adds it to the Meta-buffer only if it is sufficiently novel under the same similarity criterion, enabling continual accumulation and reuse.

Baseline	Key settings in our reproduction
CoT	Single sample; parse only `Final Answer:` field
CoT-SC	5 samples; normalize + majority vote; deterministic fallback
ToT	BFS; 5 candidates/node; top-2 expand; max depth 2; self-eval prompt
Meta-Prompting	Meta controller + experts via role prompts; iterative multi-round; experts see only isolated triple-quoted instructions ("fresh eyes"); stop on final-answer marker; no code-execution expert
Meta-Reasoning	Strategy selection; when strategy is ToT/CoT-SC use the same settings as above
TTPI	Step-stage interventions; branch selection via PPL when available (no RDS without internal-layer access)
BoT	Distill–retrieve–instantiate–update; similarity-threshold gating for template retrieval and buffer updates

Table 3: Hyperparameter and procedure summary for baseline reproductions.

C Implementation

C.1 Heuristic Construction of Instance-level Reflections

For each instance, we construct an *instance-level reflection* by aggregating signals from its interaction trace across the REASONER–MONITOR–CONTROLLER loop. The reflection is a compact, structured summary of (i) the task outcome, (ii) how many inner-loop iterations were required, and (iii) coarse-grained role-level quality ratings. It is derived only from trace-visible fields (plus the correctness label) and does not require additional model calls. We use the correctness flag only as a binary outcome tag for reflection labeling; the gold answer text $y_t^\star$ is never provided to any LLM call during reflection, distillation, retrieval, or consolidation.

Inputs extracted from the trace. Let the trace contain T iteration records $\mathcal{H} = \{h_1, \ldots, h_T\}$. From each iteration h_t, we extract: (i) the monitor's *error-found flag* e_t, (ii) the controller's *action code* a_t, and lightweight per-iteration facts for constructing each agent's *behavior trajectory* (defined below). We also use instance-level metadata: terminal status s (e.g., accepted/corrected vs. max-iteration) and correctness flag $y \in \{0, 1\}$. We denote the monitor's final-iteration flag by e_T.

We summarize two trace-level counts:

$$m_{\texttt{YES}} = \sum_{t=1}^{T} \mathbb{I}[e_t = \texttt{YES}], \qquad c_3 = \sum_{t=1}^{T} \mathbb{I}[a_t = 3], \tag{14}$$

where $m_{\texttt{YES}}$ counts how often the monitor flags an error, and c_3 counts how often the controller selects the restart action.

Structured reflection fields. Each instance-level reflection contains:

- **Task outcome:** a binary label (`success` or `failure`) derived from the correctness flag.
- **Task quality:** a three-level label (`A`/`B`/`C`) describing whether the system solved the instance cleanly in one iteration, required multiple iterations, or failed to converge.
- **Reasoner quality:** a three-level rating described as *good*, *ok*, or *poor* (stored as `R_good`, `R_ok`, `R_poor` in logs), derived from termination type, iteration count, and the prevalence of monitor-flagged errors.
- **Monitor quality:** a three-level rating described as *good*, *ok*, or *poor* (stored as `M_good`, `M_ok`, `M_poor`), based on whether the monitor's final judgment is aligned with successful termination (e.g., the final iteration should not still flag an error if the run terminates as accepted/corrected).
- **Controller quality:** a three-level rating described as *good*, *ok*, or *poor* (stored as `C_good`, `C_ok`, `C_poor`), based on whether the controller converges efficiently and whether it over-uses restart actions in non-convergent runs.
- **Role behavior trajectories:** three compact per-role trajectories extracted from the trace, each recording the key observable actions taken at every iteration (in chronological order). Concretely:
  - **Reasoner trajectory:** the sequence of extracted final answers $\{\hat{z}_t\}_{t=1}^{T}$ (and optionally a short excerpt pointer to the full response text for logging).
  - **Monitor trajectory:** the sequence $\{(e_t, \mathrm{step}_t, \mathrm{desc}_t)\}_{t=1}^{T}$, where e_t is the error-found flag and $\mathrm{step}_t, \mathrm{desc}_t$ are the reported error location and short description (when available).

– **Controller trajectory:** the sequence $\{(a_t, \text{just}_t)\}_{t=1}^{T}$, where a_t is the action code and just_t is the controller's justification string (or a brief extract).

Heuristic rules.

- **Task outcome.** We set task outcome to `success` if $y = 1$; otherwise `failure`.
- **Task quality.** We set task quality to `A` if s indicates accepted/corrected and $T = 1$, to `B` if s indicates accepted/corrected and $T > 1$, and to `C` otherwise (including max-iteration termination).
- **Reasoner quality.** We rate the reasoner as *good* if the run is accepted/corrected within two iterations; as *poor* if the run hits the maximum iteration budget and more than half of iterations are flagged as erroneous by the monitor (i.e., $m_{\texttt{YES}} > T/2$); otherwise as *ok*.
- **Monitor quality.** For accepted/corrected runs, we rate the monitor as *good* if the final iteration does not flag an error ($e_T = \texttt{NO}$), and as *poor* if it still flags an error ($e_T = \texttt{YES}$); otherwise we assign *ok*. For non-accepted runs, we assign *ok*.
- **Controller quality.** We rate the controller as *good* if the run is accepted/corrected within three iterations; as *poor* if the run hits the maximum iteration budget and more than half of iterations choose the restart action (i.e., $c_3 > T/2$); otherwise as *ok*.

Optional episode summaries (logging only). For debugging and analysis, we also produce three human-readable reflections (one per role), each including the auto-wrapping question brief and enumerating the corresponding role behavior trajectory over iterations (e.g., the reasoner's final answer per iteration; the monitor's error flags/steps/descriptions; the controller's actions and justifications). These texts are not required by the reflection schema itself and do not affect downstream construction.

C.2 MRO role strategy Update

The prompt update operator in Eq. (13) is implemented via an *LLM-based prompt composer*. Concretely, for each role (REASONER/MONITOR/CONTROLLER), we maintain a role-specific policy prompt at time t and update it by composing: (i) the previous policy prompt P_t, (ii) the current global meta-knowledge state K_b, (iii) the retrieved instance-relevant micro-lessons $\widetilde{\mathcal{L}}_t$ from Eq. (15), and (iv) the current instance text x_i. The composer itself is realized by a dedicated prompt template (see Appendix H) and is executed by calling the backbone model LLM_θ.

Template and output format. We use a fixed *prompt-composer template* defined in Appendix H that instructs the model to produce updated role prompts in a pre-specified structured schema (e.g., with explicit role fields such as `P_R`, `P_M`, `P_C`). This schema is designed so that the updated prompts can be parsed deterministically and directly fed into the next MRO call.

Validity checks, regeneration, and fallback. After generation, we apply lightweight checks to ensure the composer output is usable: (i) all required role fields are present; (ii) the output is parseable under the expected schema; and (iii) the resulting prompt length does not exceed a pre-set context budget for the next inference call. If any check fails, we re-run the prompt composer to regenerate the updated prompts . If regeneration still fails, we fall back to using the previous prompt P_t for that role (i.e., no update is applied for that step).

Trace logging. When a prompt update is attempted, we log the update metadata (whether an update was used, whether it succeeded, which model performed the update, and prompts before/after update) under the **Updated role policy** field in each role's trace record (Appendix C.5).

C.3 In-batch Distillation

The distillation operator $\text{Distill}(\cdot)$ in Eq. (10) is implemented as an *LLM-based lesson distiller*. For each role (REASONER/MONITOR/CONTROLLER), we run distillation *independently* to obtain role-specific micro-lessons. Given a batch $\mathcal{B}_b$, the distiller takes as input a set of instance-level reflections selected by the heuristic filter in Eq. (9), and outputs a batch-level micro-lesson ℓ_b that summarizes *reusable tactics* (from $\mathcal{S}_b^+$) and *failure patterns to avoid* (from $\mathcal{S}_b^-$).

Inputs. For batch $\mathcal{B}_b$, we form the distillation input set $\{r_t\}_{t \in \mathcal{S}_b^+ \cup \mathcal{S}_b^-}$, where each reflection r_t

is constructed deterministically (Appendix C.1) and contains only trace-derived fields plus a binary outcome tag. Importantly, we never provide the gold answer text $y_t^\star$ to the distiller; the correctness flag is used only as a coarse label (`success`/`failure`) inside reflections.

Prompt template and output format. We implement the distiller using a fixed prompt template (Appendix H) executed by the backbone model LLM_θ. The template specifies the target role and instructs the model to (i) identify recurring patterns across reflections, (ii) extract actionable rules/checks for the role, and (iii) summarize both positive (*do*) and negative (*avoid*) lessons grounded in the input reflections. The distiller outputs ℓ_b in a pre-specified structured schema, as defined in the prompt.

C.4 Temporal Consolidation (T-consolidation)

The consolidation operator $\mathrm{Consol}(\cdot)$ in Eq. (11) is implemented via an *LLM-based meta-knowledge consolidator*. The goal is to maintain an evolving global meta-knowledge state K_b per role, while enabling natural forgetting through the sliding window $\mathrm{Win}_w(\cdot)$.

Inputs. At the end of batch $\mathcal{B}_b$, the consolidator receives: (i) the previous global meta-knowledge K_{b-1} for the target role, and (ii) the windowed set of recent micro-lessons $\mathcal{L}_b^{(w)} \triangleq \mathrm{Win}_w(\{\ell_j\}_{j=1}^b)$. Only micro-lessons from the most recent w batches are provided, so outdated rules are implicitly deprioritized without requiring explicit timestamps.

Prompt template and output format. We implement the consolidator using a fixed prompt template (Appendix H) executed by LLM_θ. The template instructs the model to consolidate $\mathcal{L}_b^{(w)}$ into an updated global state K_b by merging redundant lessons, resolving minor inconsistencies, and rewriting the remaining rules into a concise, role-executable form. The consolidator outputs K_b in a pre-specified structured schema, as defined in the prompt, which can be directly injected into subsequent role-policy updates and guided inference.

C.5 Trace Schemas

For each input instance, we record an ordered *interaction trace* as a sequence of inner-loop iterations. Each iteration corresponds to one complete cycle of the system and contains the outputs of three roles: a REASONER (proposal), a MONITOR (audit), and a CONTROLLER (decision). The trace is designed to capture *what information is produced and exchanged* during test-time evolution.

Overall structure. A trace is a chronological list of iteration records (`history`), ordered from the earliest to the latest:

```
history: [ { ... iteration record ...
}, { ... }, . . . ]
```

Iteration record. Each iteration record contains:

- **Iteration index**: an integer indicating which inner-loop cycle this record corresponds to (1-indexed).
- **Reasoner output**: the candidate solution produced in this cycle.
- **Monitor output**: the diagnostic report produced by auditing the reasoner output.
- **Controller output**: the control decision that determines whether to accept, patch, or request a restart.

Reasoner output. The reasoner output contains:

- **Final answer**: the extracted final answer for this iteration.
- **Full response text**: the complete natural-language solution text produced by the reasoner (including intermediate steps).
- **Updated role policy**: optional metadata describing a policy update applied to the reasoner for this iteration, including whether an update was used, whether it succeeded, which model performed the update, and the prompts before/after the update. (See Appendix C.2 for how the updated prompts are composed.)

Monitor output. The monitor output contains:

- **Error found flag**: a boolean indicating whether the monitor identified an error that could affect the final answer.
- **Error location**: an index or textual pointer indicating where the first major error was detected (or `NONE` if no error is found).

- **Error description**: a short actionable description of the detected issue.
- **Audit explanation**: a natural-language justification of the monitor's judgment.
- **Full report text**: the raw monitor report string, typically containing both the explanation and a structured summary.
- **Updated role policy**: optional metadata describing a policy update applied to the monitor for this iteration. (See Appendix C.2.)

Controller output. The controller output contains:

- **Action**: a discrete decision code indicating how the system proceeds. We use three actions:
  - **Accept**: accept the current iteration's answer and terminate the inner loop.
  - **Patch**: provide a corrected reasoning (if available) and a corrected final answer, then terminate the inner loop.
  - **Restart**: reject the current attempt and provide high-level revision suggestions for the next iteration.

  For logging convenience, we encode actions as integers: ACCEPT= 1, PATCH= 2, RESTART= 3.
- **Justification**: a short explanation for why the controller selected the action.
- **Revision suggestions**: when the action is *restart*, a brief action-oriented plan for the reasoner to attempt next.
- **Corrected reasoning**: when the action is *patch*, the controller's corrected reasoning text.
- **Final answer**: when the action is *accept* or *patch*, the accepted/patched final answer; otherwise empty.
- **Full decision text**: the raw controller decision string, typically including both justification and a structured summary.
- **Updated role policy**: optional metadata describing a policy update applied to the controller for this iteration. (See Appendix C.2.)

Termination condition. An instance trace ends when the controller emits *accept* or *patch*, or when a maximum iteration budget is reached (in which case the last record is typically a *restart*). The resulting `history` list thus provides a complete chronological record of proposals, audits, and control decisions for the instance.

C.6 Retrieval Settings for In-batch Lesson Conditioning

For each sample x_i in the next batch, we first retrieve up to k most relevant lessons from the lesson buffer restricted to a sliding window of size w:

$$\widetilde{\mathcal{L}}_t = \mathrm{Retrieve}\Big(x_i,\ \mathcal{L}_b^{(w)} \mid k\Big), \qquad (15)$$

where $\mathcal{L}_b^{(w)}$ denotes the subset of buffered lessons that fall inside the most recent window of length w (i.e., the last w batches/steps, depending on the buffer implementation). If fewer than k lessons are available in the window, we return all available lessons.

Similarity function. We perform retrieval by cosine similarity between the representation of the current problem and that of each candidate lesson. We obtain embeddings using **text-embedding-qwen3-embedding-0.6b**. Concretely, let $\mathbf{e}(x_i)$ be the embedding of the problem text x_i, and let $\mathbf{e}(\ell)$ be the embedding of a lesson $\ell \in \mathcal{L}_b^{(w)}$ (computed from the lesson text in its serialized form, i.e., concatenating its main fields such as `trigger` and `action`). The cosine similarity is:

$$s(x_i, \ell) = \frac{\mathbf{e}(x_i)^\top \mathbf{e}(\ell)}{\|\mathbf{e}(x_i)\|_2 \, \|\mathbf{e}(\ell)\|_2}. \qquad (16)$$

The retrieval operator $\mathrm{Retrieve}(\cdot)$ ranks candidates in $\mathcal{L}_b^{(w)}$ by $s(x_i, \ell)$ and returns the top-k lessons.

Hyperparameters. Unless otherwise stated, we use:

- **Top-k retrieval:** $k = 3$.
- **Window size:** $w = 3$.
- **Embedding model: text-embedding-qwen3-embedding-0.6b**.
- **Metric:** cosine similarity as in Eq. (16).

Table 4: Batch size configuration and resulting number of batches for each dataset.

Dataset	Dataset size N	Batch size B	#Batches M
MATH500	500	50	10
GSM8K	1319	100	14
TheoremQA	800	80	10
Game of 24	100	10	10

C.7 Key Hyperparameters and Runtime Budget

C.7.1 Batch Size

We evaluate on four datasets: MATH500 (500 problems), GSM8K (1319 problems), TheoremQA (800 problems), and Game of 24 (100 problems). For each dataset, the batch size is set to one-tenth of the dataset size, with a lower bound of 10 and an upper bound of 100.

Let N denote the number of instances in a dataset. The batch size B is computed as:

$$B = \min\left(100,\ \max\left(10,\ \left\lfloor \frac{N}{10} \right\rfloor\right)\right). \quad (17)$$

The number of batches M for a dataset is:

$$M = \left\lceil \frac{N}{B} \right\rceil. \quad (18)$$

C.7.2 Backbone Models

API-based models. **GPT-4o-mini** is decoded using the provider's default decoding parameters (i.e., we do not override sampling or penalty parameters). **Gemini-2.0-flash** is also decoded with the provider's default generation configuration.

Open-source models. We evaluate two open-source backbones: **Qwen3-8B** and **Llama-3.1-8B-Instruct**. For **Qwen3-8B**, the thinking mode is disabled; otherwise, decoding uses default settings (no additional manual tuning). For **Llama-3.1-8B-Instruct**, all decoding parameters remain at default values.

No explicit output-token cap. Across *all* models and methods, we do not impose an explicit upper bound on output tokens (e.g., we do not set `max_tokens`/`max_output_tokens`/`max_new_tokens` at the experiment level). Generation terminates naturally (e.g., by end-of-sequence) or by provider-enforced limits.

Table 5: Effect of `maxiteration` on accuracy and average decoding tokens (MATH500, `gpt-4o-mini`).

`maxiteration`	Accuracy (%)	#Tokens
2	84.6	1430.36
3	85.8	1576.24
4	83.2	1707.99
5	82.4	1861.50
6	84.4	1945.70

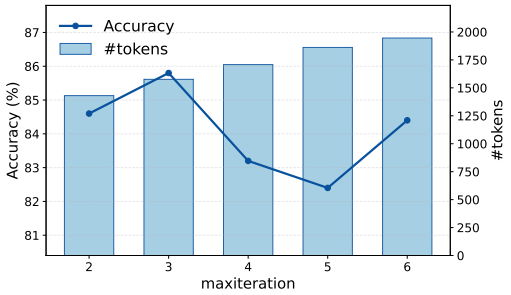

Figure 5: Iteration budget trade-off on MATH500 with `gpt-4o-mini`: accuracy (line) versus average decoding tokens (bars) under different `maxiteration`.

D Iteration Trade-off Research

We study the iteration–cost trade-off by varying the maximum iteration budget in the inner loop. We run `gpt-4o-mini` on MATH500 using the same pipeline as our full method, with the *only* difference being the setting of `maxiteration`. For each `maxiteration` value, we conduct one full evaluation and report accuracy (%) and the average number of decoding tokens per instance.

Discussion. A smaller `maxiteration` can be *too short* for hard instances. With limited inner-loop cycles, the system may terminate before the reasoner has enough opportunities to incorporate monitor feedback and controller guidance. As a result, correctable mistakes may remain unresolved (or the controller may be forced into premature restart/accept decisions), typically reducing accuracy even though fewer iterations save tokens.

Conversely, a larger `maxiteration` can be *too long*. Increasing the iteration budget monotonically increases generation cost because each additional cycle incurs extra proposals, audits, and decisions. In addition, allowing many iterations can introduce over-refinement effects: later cycles may over-correct a nearly-correct solution, drift away from a good partial derivation, or accumulate inconsistencies across successive revisions. This can produce diminishing (or even negative)

returns in accuracy while token usage continues to rise.

Given these trade-offs, using a small number of iterations is desirable for efficiency, but it must remain sufficient for feedback-driven correction. In our setting, `maxiteration=3` offers a reasonable balance: it enables an extra round of revision beyond a short budget, while avoiding the heavier cost and potential over-refinement associated with longer iteration limits.

E Does MC^2's role-strategy update inject task-relevant Reasoner guidance?

This experiment evaluates whether the *role-strategy update* in MC^2 (i.e., updating the Reasoner policy prompt via retrieved micro-lessons and consolidated meta-knowledge; cf. the *update MRO role strategy* step) injects *task-relevant* guidance for the current instance. Importantly, we aim to measure relevance beyond trivial prompt overlap with the problem statement.

Setup. We evaluate on **MATH500** using the **GPT-4o-mini** backbone. Following Section C.7.1, the batch size is $B{=}50$ (thus $M{=}10$ batches). For each instance i in batch b, we take the Reasoner's *compiled* prompt after the role-strategy update (the rewritten/augmented prompt used to call the Reasoner) and explicitly remove the full problem text from it, yielding a *guidance-only* snippet:

$$g_i \;=\; \texttt{FinalPrompt}_i \;-\; \texttt{Question}_i.$$

We then compute the cosine similarity between the instance question and this guidance-only snippet using the same embedding model as in our retrieval component:

$$s_i \;=\; \cos\big(\mathrm{Emb}(\texttt{Question}_i),\; \mathrm{Emb}(g_i)\big).$$

As a within-batch control baseline, we randomly sample a guidance-only snippet g_j from *the same batch* ($j \neq i$) and compute:

$$s_i^{\mathrm{rand}} \;=\; \cos\big(\mathrm{Emb}(\texttt{Question}_i),\; \mathrm{Emb}(g_j)\big).$$

For multi-turn instances, we compute similarities per turn and report their mean. Finally, for each batch b, we report the mean similarity over instances in that batch. Because batch 1 has no prior accumulated experience to retrieve/compile into the role strategy, we report batches 2–10.

Agent	p_{keyops}
Reasoner	0.9146
Monitor	0.9618
Controller	0.9848

Table 6: GPT-5 binary-judge evaluation of whether the agent's updated role policies contains task-specific key operations/checkpoints that would materially help solve the given question. Results are averaged per instance (and across turns for multi-turn instances), then averaged over instances.

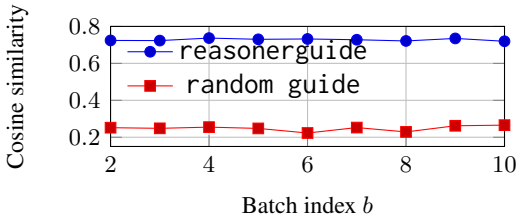

Figure 6: Embedding cosine similarity between each question and the *guidance-only* snippet produced by the MC^2 role-strategy update for the Reasoner (question text removed), compared with a within-batch random baseline. Results are averaged per batch ($B{=}50$) on MATH500 using GPT-4o-mini.

What this shows. Because the question text is explicitly removed from the updated prompt, high similarity cannot be explained by copying or restating the problem. A consistently higher `reasonerguide` similarity than the within-batch `random guide` baseline indicates that the *role-strategy update* injects *instance-aligned* guidance (i.e., cues that are specific to the current problem) rather than generic advice. The stable gap across batches further supports that the MC^2 update mechanism reliably compiles accumulated experience into targeted Reasoner guidance as meta-knowledge grows.

LLM-as-judge corroboration. As a complementary check, we use **GPT-5** as a binary judge to evaluate whether the updated role policies contains *task-specific key operations/checkpoints* that would materially help solve the given question. The judge is prompted with the instance question and the agent's updated role policies, and outputs a strict binary decision per turn; for multi-turn instances we average across turns and then report the per-agent mean.

Table 7: Order sensitivity under sliding-window meta-knowledge consolidation. Δ is computed as `Shuffled` $-$ `Default` and shown inline in the shuffled column.

	Default order	Shuffled order
	gpt-4o-mini	
math500	85.13	85.06 (-0.07)
game-of-24	88.67	87.33 (-1.34)
	gemini-2.0-flash	
math500	96.73	96.93 ($+0.20$)
game-of-24	94.33	93.67 (-0.66)

F Does data order affect the consolidated global meta-knowledge?

Our global meta-knowledge is updated via a bounded sliding-window consolidation mechanism (Eq. (11)), which raises a natural question: *does the presentation order of input instances affect the resulting meta-knowledge and final performance?* To test order sensitivity, we run MC^2 with two API backbones, `gpt-4o-mini` and `gemini-2.0-flash`, on two datasets, GAME OF 24 and MATH500. We compare the dataset's released instance order (i.e., the original file order used in our data loader) with a randomly permuted order (shuffled instances) under the same hyperparameters and evaluation protocol. For the shuffled setting, each run uses a different random seed for permutation. Overall, we observe that shuffling the dataset order has negligible impact on final performance, suggesting that the sliding-window consolidation is reasonably robust to instance ordering.

G Example

This section presents one concrete MC^2 episode as four single-column card figures. The episode (Geometry, Index 400) completes in two MRO inner iterations: the first attempt outputs an incorrect answer (7) due to a mis-modeled segment relationship; the monitor flags the error (error_step=4) and the controller triggers a restart; the second attempt uses coordinate placement and a center-distance tangency equation to obtain the correct radius $\frac{14}{3}$.

Card 1Instance Overview

Index: 400
Subject: Geometry
Question: In the circle with center Q, radii AQ and BQ form a right angle. The two smaller regions are tangent semicircles. The radius of the circle with center Q is 14 inches. Find the radius of the smaller semicircle.
Gold answer: $\frac{14}{3}$
Model final: $\frac{14}{3}$
Status: accepted
Iterations: 2
Rank: `task_quality=B; task_outcome=success; reasoner=R_good; monitor=M_ok; controller=C_good`

Figure 7: Card 1: Instance overview (Index 400).

Card 2Trajectory at a Glance

Iter 1
- **Reasoner answer**: 7
- **Monitor**: `error_found=YES`, `error_step=4`
- **Monitor note**: The equation $2r = 28$ is incorrect (segment relationship is mis-modeled).
- **Controller**: `action=RESTART (Action 3)`
- **Justification**: Major geometric relationship issues; restart with correct configuration.
- **Outcome**: continue

Iter 2
- **Reasoner answer**: $\frac{14}{3}$
- **Monitor**: `error_found=NO`
- **Monitor note**: All steps are consistent and correct.
- **Controller**: `action=ACCEPT (Action 1)`
- **Outcome**: terminate

Figure 8: Card 2: MRO trajectory at a glance.

Card 3Iteration 1 (Failure)

Reasoner final_answer: 7
Core issue (summary): Unjustified segment equality assumptions lead to an invalid equation for r.
Monitor: `error_found=YES`, `error_step=4`
Monitor description: The equation $2r = 28$ neglects the actual placement/tangency constraints of the semicircles.
Controller decision: RESTART
Controller suggestion: Redefine centers and use tangency as a distance constraint (center distance = sum of radii).

Figure 9: Card 3: Failure iteration and diagnosis.

Card 4Iteration 2 (Success)

Setup
$Q = (0, 0)$, $A = (14, 0)$, $B = (0, 14)$.
Semicircle on diameter AQ: radius 7, center $C = (7, 0)$.
Smaller semicircle radius r, internally tangent to the big circle: center $D = (0, 14 - r)$.

Tangency constraint (two semicircles tangent)

$$\sqrt{(7-0)^2 + (0-(14-r))^2} = 7 + r.$$

Solve

$$\begin{aligned} 49 + (14 - r)^2 &= (7 + r)^2 \\ 196 &= 42r \\ r &= \frac{14}{3}. \end{aligned}$$

Final answer: $\boxed{\frac{14}{3}}$

Figure 10: Card 4: Successful iteration and solution.

H Prompt

We provide the full prompt templates used by the Reasoner, Monitor, and Controller in MC^2, covering cold-start inference, inference with retrieved micro-lessons/meta-knowledge, role-policy updates, reflection-to-micro-lesson distillation, and windowed meta-knowledge consolidation.

Reasoner(No micro-lessons or metaknowledge)

```
You are a reasoning expert.
Problem:
{problem}
Please think step by step about how to solve this problem.
Enclose your final answer in the form:
\\boxed{{your_answer_here}}.
```

Figure 11: Reasoner prompt (cold start; no micro-lessons or meta-knowledge).

Reasoner(There are micro-lessons or metaknowledge)

```
You are a reasoning expert.
Before solving, internalize the following task-specific
guidance:
{guidance}
Now solve the problem step by step.
Problem:
{problem}
Show your full reasoning. Enclose your final answer in
the form: \\boxed{{your_answer_here}}.
```

Figure 12: Reasoner prompt (with retrieved micro-lessons / meta-knowledge guidance).

Reasoner role policy updates

```
You are a meta-level coach helping the Reasoner plan how to solve one problem.
    You will receive:
    - The problem statement.
    - Optional recent feedback from the Monitor/Controller about a previous attempt.
    - Global meta-knowledge for the Reasoner (cross-task lessons).
    - Retrieved task-level micro-lessons (similar prior cases).
    ---------- Problem ----------
    {question}
    ---------- Recent feedback (may be empty) ----------
    {recent_feedback}
    ---------- Reasoner meta-knowledge (may be empty) ----------
    {meta_knowledge}
    ---------- Retrieved micro-lessons (may be empty) ----------
    {micro_lessons}
     Your job:
    - Read everything carefully.
    - Produce a concise guidance note telling the Reasoner how to approach THIS problem.

    The guidance must:
    - Focus on high-level reasoning strategy, not detailed algebra steps.
    - Mention sub-goals, useful checks, and common pitfalls to avoid.
    - Be specific to the problem structure, but NOT repeat the whole problem text.
    - Be at most 6 bullet points, each no more than 20 words.
    Format your output exactly as:
    Guidance:
    - ...
    - ...
    (only bullet lines after "Guidance:"; no extra text)
```

Figure 13: Reasoner role policy update prompt (meta-level coaching to generate task-specific guidance).

Reasoner reflection distillation into micro-lesson

```
You are writing a task-specific execution exemplar for a Reasoner.
You will be given:
- The original question.
- The Reasoner's final reasoning output (may be imperfect).
- The Monitor's last feedback and the Controller's last decision.
Goal:
Produce ONE micro-lesson that is useful as a concrete example of "how to execute" on THIS
problem.
It may reference the problem content and numbers. That is allowed and encouraged.
Also include a short descriptor so it can be retrieved for similar future tasks.
Output format (STRICT):
Descriptor: <one line, abstract retrieval key, NO numbers>
Applicable_when: <one line, abstract condition, NO numbers>
Execution_recipe:
  1) <imperative step>  2) <imperative step>
  3) <imperative step>  4) <optional>
Key_checks:
  - <check>
  - <check>
Worked_snippet:
  <3-10 lines showing a representative fragment for THIS task; numbers allowed>
Failure_mode_to_avoid:
  - <one line>
Rules:
- Keep it short and operational; prefer imperative steps.
- Do NOT include advice that changes answer formatting requirements.
### Question
{question}
### Reasoner final output
{reasoner_output}
### Monitor feedback (last)
{monitor_feedback}
### Controller decision (last)
{controller_decision}
### Output the Micro-Lesson:
```

Figure 14: Reasoner reflection distillation prompt for micro-lesson (execution exemplar with checks and failure modes).

Reasoner Windowed Meta-Knowledge Consolidation

```
You maintain the Reasoner's long-term meta-knowledge.
Inputs:
- Previous meta-knowledge (stable rules).
- Recent micro-lessons (task-specific exemplars with descriptors).
Task:
Update the meta-knowledge by:
1) Preserving good existing rules (do NOT rewrite for style).
2) Merging duplicates.
3) Adding at most 2 new rules only if strongly supported by multiple lessons.
4) Deleting rules only if contradicted or consistently harmful.
Output format (STRICT):
- EXACTLY 6 rules.
- Each rule MUST have a stable ID and be one sentence.
- Use this format exactly:
  R1: ...
  R2: ...
  R3: ...
  R4: ...
  R5: ...
  R6: ...
Constraints:
- No task references, no numbers, no examples.
- Each rule must be actionable and checkable (avoid vague phrases).
- Prefer "Do X before Y" / "If condition then action" style.
### Previous Meta-Knowledge
{previous_meta_knowledge}
### Recent Micro-Lessons
{recent_micro_lessons}
### Output updated Meta-Knowledge:
```

Figure 15: Reasoner prompt for windowed meta-knowledge consolidation (merge and update stable rules from recent micro-lessons).

Monitor(No micro-lessons or metaknowledge)

```
You are a reasoning monitor. Your task is to carefully review the
reasoning trajectory and judge whether any step contains a mistake
that could affect the final answer.
The following is a {subject} reasoning problem:
{problem}
Here is the reasoning trajectory:
{trajectory}
Please read it carefully and analyze whether any step contains a logical
flaw, incorrect assumption, computational error, or unjustified jump.
Your response should contain two parts:
1. A natural-language explanation of your analysis.
2. A short summary section in the following fixed format:
Error_found: YES or NO
Error_step: <step number or NONE>
Error_description: <ONE sentences that briefly state which stage of
the solution is unreliable>
Write your explanation naturally, but strictly follow the summary
format at the end.
```

Figure 16: Monitor prompt (cold start; no micro-lessons or meta-knowledge).

Monitor(There are micro-lessons or metaknowledge)

You are a reasoning monitor. Your task is to carefully review the reasoning trajectory and judge whether any step contains a mistake that could affect the final answer.
Problem:
{problem}
Reasoning trajectory:
{trajectory}
Before reviewing, study the following task-specific review focus:
{guidance}

Now analyze the trajectory carefully.
Your response should contain two parts:
1. A natural-language explanation of your analysis.
2. A short summary section in the following fixed format:
Error_found: YES or NO
Error_step: <step number or NONE>
Error_description: <ONE sentences that briefly state which stage of the solution is unreliable>

Write your explanation naturally, but strictly follow the summary format at the end.

Figure 17: Monitor prompt (with retrieved micro-lessons / meta-knowledge review focus).

Monitor role policy updates

You are a meta-level coach helping the Monitor agent decide how to review a trajectory.
You will receive:
- The problem statement.
- The current reasoning trajectory.
- Global meta-knowledge for the Monitor.
- Retrieved micro-lessons about monitoring.
Your job:
- Produce a short review plan describing what to focus on for THIS trajectory.
The plan must:
- Highlight what types of errors are likely (logical gaps, arithmetic, conditions, etc.).
- Indicate where to be especially strict or cautious.
- Be at most 5 bullet points, each no more than 20 words.
Format your output exactly as:
Review plan:
- ...
- ...
---------- Problem ----------
{question}
---------- Reasoning trajectory (truncated) ----------
{trajectory}
---------- Monitor meta-knowledge (may be empty) ----------
{meta_knowledge}
---------- Retrieved micro-lessons (may be empty) ----------
{micro_lessons}

Figure 18: Monitor role policy update prompt (meta-level coaching to produce a concise review plan).

Monitor reflection distillation into micro-lesson

```
You are writing a task-specific diagnostic exemplar for a Monitor.
Goal:
Create ONE micro-lesson demonstrating how to detect the key issue(s) in THIS trajectory on THIS
task.
It may reference the trajectory content and numbers. That is allowed.
Also include a short descriptor for retrieval.
Output format (STRICT):
Descriptor: <abstract retrieval key, NO numbers>
Red_flags_seen:
  - <red flag>
  - <red flag>
Where_to_check_first:
  - <location heuristic>
Diagnostic_procedure:
  1) <step>
  2) <step>
  3) <step>
Minimal_countercheck:
  <one concrete check that would reveal the issue on THIS task; numbers allowed>
Good_error_report_example:
  Error_found: YES or NO
  Error_step: <step number or NONE>
  Error_description: <ONE sentence>
Rules:
- Focus on detection and pinpointing, not solving the problem.
- Do NOT change the Monitor's required summary schema in real runs.
### Question
{question}
### Reasoning trajectory
{trajectory}
### Monitor final output
{monitor_output}
### Controller decision (last)
{controller_decision}
### Output the Micro-Lesson:
```

Figure 19: Monitor reflection distillation prompt for micro-lesson (diagnostic exemplar with red flags and minimal countercheck).

Monitor Windowed Meta-Knowledge Consolidation

```
You maintain the Monitor’s long-term diagnostic meta-knowledge.
Inputs:
- Previous meta-knowledge (stable diagnostic rules).
- Recent micro-lessons (task-specific diagnostic exemplars with descriptors).
Task:
Update the meta-knowledge by:
1) Preserving good existing rules (do NOT rewrite for style).
2) Merging duplicates.
3) Adding at most 2 new rules only if strongly supported by multiple lessons.
4) Deleting rules only if contradicted or consistently harmful.
Output format (STRICT):
- EXACTLY 6 rules.
- Each rule MUST have a stable ID and be one sentence:
  M1: ...
  M2: ...
  M3: ...
  M4: ...
  M5: ...
  M6: ...
Constraints:
- No task references, no numbers, no examples.
- Rules must be diagnostic (how to detect/locate issues), not solving tips.
- Make each rule operational and checkable.
### Previous Meta-Knowledge
{previous_meta_knowledge}
### Recent Micro-Lessons
{recent_micro_lessons}
### Output updated Meta-Knowledge:
```

Figure 20: Monitor prompt for windowed meta-knowledge consolidation (update long-term diagnostic rules from recent micro-lessons).

Controller(No micro-lessons or metaknowledge)

```
You are the controller of a multi-agent reasoning system. Your job is to decide how to proceed
based on the reasoning trajectory and the monitoring analysis.
Here is the {subject} reasoning problem:
{problem}
Here is the reasoning trajectory:
{trajectory}
Here is the monitor's analysis:
{monitor_reason}
You must choose one of the following three actions:
Action 1 — The reasoning has no meaningful issues. Accept the trajectory and final answer.
Action 2 — The reasoning contains MINOR issues. You should directly revise and fix the
trajectory yourself, then provide the corrected reasoning and final answer.
Action 3 — The reasoning contains MAJOR issues. Do not solve the problem yourself. Provide
clear guidance on how the reasoner should revise the reasoning.
Your response should contain:
1. A brief natural-language justification.
2. A summary section in the following fixed format:
Action: 1 or 2 or 3
Justification: <short explanation>
If Action = 1:
   Final_answer: ### your_answer_here ###
If Action = 2:
   Corrected_reasoning: <your improved reasoning>
   Final_answer: ### your_answer_here ###
If Action = 3:
   Suggestions:
      - Give a short, high-level plan for re-solving the problem from scratch in ONE sentence.
        Do NOT copy any intermediate expressions from the trajectory.
        Keep the suggestions abstract and action-oriented.
Write naturally, but strictly follow the summary fields and their labels.
```

Figure 21: Controller prompt (cold start; no micro-lessons or meta-knowledge).

Controller(There are micro-lessons or metaknowledge)

```
You are the controller of a multi-agent reasoning system.
Problem:
{problem}
Current reasoning trajectory:
{trajectory}
Monitor's analysis:
{monitor_reason}
Before deciding, internalize this task-specific decision policy:
{guidance}
You must choose one of the following three actions:
Action 1 — The reasoning has no meaningful issues. Accept the trajectory and final answer.
Action 2 — The reasoning contains MINOR issues. You should directly revise and fix the trajectory yourself,
        then provide the corrected reasoning and final answer.
Action 3 — The reasoning contains MAJOR issues. Do not solve the problem yourself. Provide clear
guidance
        on how the reasoner should revise the reasoning.
Your response should contain:
1. A brief natural-language justification.
2. A summary section in the following fixed format:
Action: 1 or 2 or 3
Justification: <short explanation>
If Action = 1:
   Final_answer: ### your_answer_here ###
If Action = 2:
   Corrected_reasoning: <your improved reasoning>
   Final_answer: ### your_answer_here ###
If Action = 3:
   Suggestions:
      - Give a short, high-level plan for re-solving the problem from scratch in ONE sentence.
        Do NOT copy any intermediate expressions from the trajectory. Keep the suggestions abstract and
action-oriented.

Write naturally, but strictly follow the summary fields and their labels.
```

Figure 22: Controller prompt (with retrieved micro-lessons / meta-knowledge decision policy).

Controller role policy updates

```
You are a meta-level coach helping the Controller choose between Action 1/2/3.
    You will receive:
    - The problem statement.
    - The current reasoning trajectory.
    - The Monitor's analysis.
    - Global meta-knowledge for the Controller.
    - Retrieved micro-lessons about control decisions.
    Your job:
    - Produce a short decision policy tailored to THIS situation.
    The policy must:
    - Explain what signals should lead to Action 1, 2, or 3.
    - Emphasize how to use Monitor warnings and trajectory stability.
    - Be at most 5 bullet points, each no more than 20 words.
    Format your output exactly as:
    Decision policy:
    - ...
    - ...
    ---------- Problem ----------
    {question}
    ---------- Reasoning trajectory (truncated) ----------
    {trajectory}
    ---------- Monitor analysis (truncated) ----------
    {monitor_reason}
    ---------- Controller meta-knowledge (may be empty) ----------
    {meta_knowledge}
    ---------- Retrieved micro-lessons (may be empty) ----------
    {micro_lessons}
```

Figure 23: Controller role policy update prompt (meta-level coaching to produce a concise decision policy).

Controller reflection distillation into micro-lesson

```
You are writing a task-specific control exemplar for a Controller.
Goal:
Create ONE micro-lesson showing how to choose Action 1/2/3 on THIS case, based on signals.
It may reference the specific signals and trajectory content.
Also include a descriptor for retrieval.
Output format (STRICT):
Descriptor: <abstract retrieval key, NO numbers>
Signals_observed:
  - <signal>
  - <signal>
Decision_rule_used:
  IF <condition> THEN Action <1/2/3> BECAUSE <reason>.
Action_taken: <1/2/3>
If_Action_3_guidance_example:
  - <ONE sentence, abstract, do NOT copy intermediate expressions>
Rules:
- Do not solve the task. Focus on routing policy and guidance quality.
- Keep Action-3 guidance abstract and non-leaky.
### Question
{question}
### Reasoning trajectory
{trajectory}
### Monitor analysis
{monitor_feedback}
### Controller final output
{controller_output}
### Output the Micro-Lesson:
```

Figure 24: Controller reflection distillation prompt for micro-lesson (control exemplar mapping signals to Action choices).

Controller Windowed Meta-Knowledge Consolidation

```
You maintain the Controller's long-term control-policy meta-knowledge.
Inputs:
- Previous meta-knowledge (stable routing rules).
- Recent micro-lessons (task-specific control exemplars with descriptors).
Task:
Update the meta-knowledge by:
1) Preserving good existing rules (do NOT rewrite for style).
2) Merging duplicates.
3) Adding at most 2 new rules only if strongly supported by multiple lessons.
4) Deleting rules only if contradicted or consistently harmful.
Output format (STRICT):
- EXACTLY 6 rules.
- Each rule MUST have a stable ID and be one sentence:
  C1: ...
  C2: ...
  C3: ...
  C4: ...
  C5: ...
  C6: ...
Constraints:
- No task references, no numbers, no examples.
- Rules must map signals -> actions (accept / revise / restart).
- Make each rule specific enough to apply.
### Previous Meta-Knowledge
{previous_meta_knowledge}
### Recent Micro-Lessons
{recent_micro_lessons}
### Output updated Meta-Knowledge:
```

Figure 25: Controller prompt for windowed meta-knowledge consolidation (update long-term control-policy rules from recent micro-lessons).