

TTML: Test-Time Memory Learning for Label-Free RLVR

Yongshi Ye^{1,3,*†}, Biao Fu^{2,*}, Xinjie Chen^{2,*}, Minpeng Liao^{2,‡}, Xiaodong Shi^{1,3,‡}

¹Institute of Artificial Intelligence, Xiamen University

²Token Foundry, Alibaba Group

³Key Laboratory of Digital Protection and Intelligent Processing of Intangible Cultural Heritage of Fujian and Taiwan (Xiamen University), Ministry of Culture and Tourism

yeyongshi@stu.xmu.edu.cn, mandel@xmu.edu.cn

Abstract

Reinforcement learning with verifiable rewards (RLVR) improves the reasoning ability of large language models (LLMs), but its standard form relies on ground-truth answers or external verifiers that are often unavailable in unlabeled test-time settings. Label-free RLVR therefore estimates pseudo rewards from model outputs. Existing methods typically use these pseudo rewards to optimize either answers or model parameters. The former cannot preserve feedback across queries, while the latter risks committing noisy signals to model parameters. This raises a central question: what should label-free rewards optimize? We propose Test-Time Memory Learning (TTML), a label-free RLVR framework that shifts the optimization target from answers or parameters to an external memory bank. TTML consists of three components: memory policy rollout selects useful memories and executes memory-augmented generation for the current query, Bayesian value update turns consensus rewards into memory value updates, and memory bank expansion extracts reusable experience from test-time trajectories. Across six benchmarks and three backbone models, TTML achieves the best Avg@8 performance over strong label-free baselines. Empirical analyses further suggest that converting label-free feedback into revisable memory values and reusable memory entries is more effective overall than using it only for single-query selection or direct parameter updates.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) has been shown to improve LLM reasoning on complex tasks such as mathematical reasoning and code generation (Jaech et al., 2024; DeepSeek-AI et al., 2025; Yu et al., 2025). However, it typically relies on human-annotated ground-

truth answers to construct verifiable rewards, which are often unavailable in unlabeled test-time settings (Ouyang et al., 2022; Shao et al., 2024; Yue et al., 2025). To reduce this dependence, existing label-free RLVR methods estimate pseudo rewards from model outputs, for example by using majority voting over sampled trajectories (Prabhudesai et al., 2025; Zhao et al., 2025; Shafayat et al., 2025; Zhang et al., 2026; Ye et al., 2026c).

A key but often overlooked question is what these pseudo signals should optimize. Some works use pseudo signals only within the current query (Wang et al., 2023; Kang et al., 2025; Madaan et al., 2023; Agarwal et al., 2025), for example using confidence to reweight sampled answers and improve answer selection. However, such gains remain confined to the current query, as costly sampled trajectories are discarded rather than extracted into reusable experience for future tasks (e.g., solution patterns). Other works instead use these signals to update model parameters, making the induced improvements persistent across queries (Prasad et al., 2025; Zuo et al., 2025; Zhang et al., 2025; Prabhudesai et al., 2025; Zhao et al., 2025). Yet this persistence comes at a cost, as directly using pseudo signals for parameter updates can lead to performance collapse or consensus traps (He et al., 2026; Pan et al., 2026; Du et al., 2026). The core challenge is finding an optimization target that preserves useful label-free feedback across queries without writing noisy signals to model parameters.

To address this, we propose the **Test-Time Memory Learning (TTML)** framework that shifts the optimization target from the current answer or model parameters to memory value. TTML keeps the base LLM frozen and adapts an external memory bank through three modules. First, **Memory Policy Rollout** retrieves candidate memories and selects useful ones by combining semantic relevance with learned memory value, then uses them to guide trajectory sampling (Section 3.2).

* Equal contribution.

† Internship work at Token Foundry, Alibaba Group.

‡ Corresponding authors.

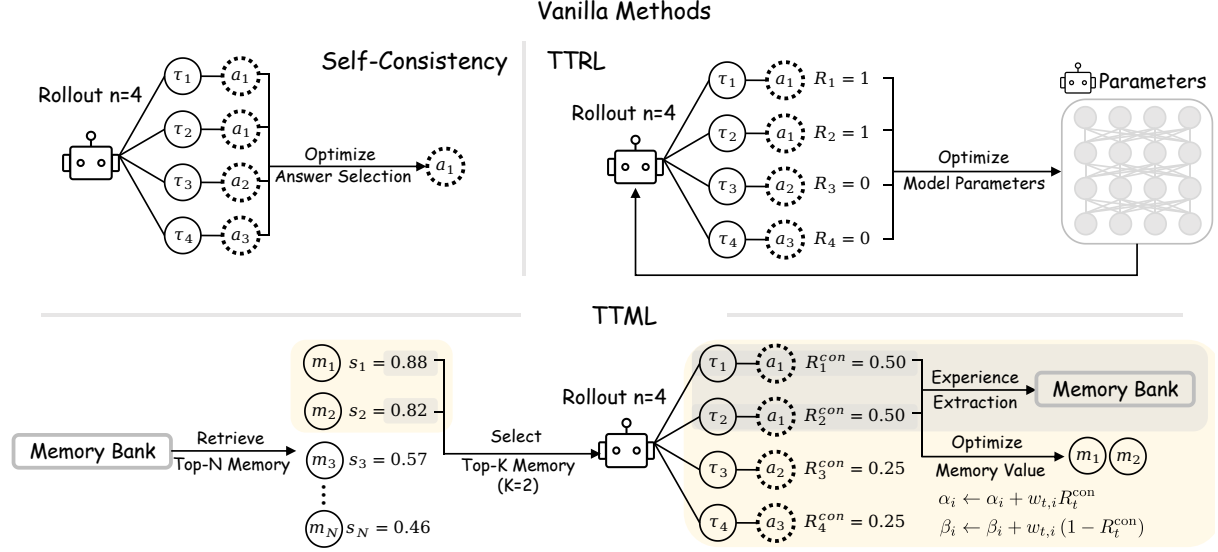


Figure 1: Comparison between vanilla label-free methods and TTML. **Top:** Self-Consistency uses sampled trajectories for answer selection, while TTRL uses majority-vote rewards to update model parameters. **Bottom:** TTML retrieves and selects Top- K memories based on semantic relevance and memory value, then uses consensus rewards from frozen-LLM rollouts to update memory values and extract reusable experience into the memory bank.

Second, **Bayesian Value Update** converts answer consensus into a label-free feedback signal and updates only the value estimates of selected memories through soft credit assignment (Section 3.3). Third, **Memory Bank Expansion** extracts new memory candidates from the majority trajectory set and merges or inserts them into the memory bank (Section 3.4). In this way, TTML turns costly test-time exploration into reusable experience, while confining noisy feedback to revisable memory records rather than global model parameters.

Our contributions are threefold:

- We identify target selection as an underexplored challenge in label-free RLVR, requiring useful feedback to persist across queries without irreversible parameter updates.
- We propose TTML, a Test-Time Memory Learning framework that resolves this challenge by converting label-free feedback into memory value updates and memory expansion, thereby accumulating reusable experience without updating the base LLM.
- Experiments across six benchmarks and three backbone models show that TTML achieves higher average performance than strong label-free baselines without updating the base model.

2 Related Work

Existing label-free RLVR methods construct intrinsic signals from model outputs when ground-truth

labels are unavailable. The key difference lies in what these signals optimize. Training-free methods optimize only the response to the current query: self-consistency and self-confidence variants select among sampled trajectories (Wang et al., 2023; Kang et al., 2025), while Self-Refine and EM-INF revise or steer the current output without updating parameters (Madaan et al., 2023; Agarwal et al., 2025). Yet these improvements do not carry beyond the current query, as sampled trajectories are discarded rather than accumulated as reusable experience. Training-based methods address this by using intrinsic signals to update model parameters (Prasad et al., 2025; Zuo et al., 2025; Zhang et al., 2025; Prabhudesai et al., 2025; Zhao et al., 2025; Ye et al., 2026c; Chen et al., 2026a,b). However, writing noisy, unverified rewards into global parameters risks capability degradation (He et al., 2026; Pan et al., 2026; Du et al., 2026). We therefore optimize memory (Ye et al., 2026a; Ouyang et al., 2026; Ye et al., 2026b): memory makes explored trajectories reusable across queries, while memory-level updates keep noisy intrinsic feedback out of the global model.

3 Method: TTML

We propose a Test-Time Memory Learning framework that keeps the LLM frozen and adapts an external memory bank through memory value updates and memory expansion. The core idea is to convert

test-time reasoning trajectories from one-off computational traces into reusable experience, and to estimate the reliability of each memory through a Bayesian value posterior for future similar queries.

3.1 Task Setup

At step t , TTML receives a test-time query x_t . The LLM is frozen; adaptation occurs through an external memory bank $\mathcal{M}_t = \{m_i\}_{i=1}^{N_t}$, initialized as $\mathcal{M}_1 = \emptyset$. Each memory is represented as

$$m_i = (k_i, c_i, \alpha_i, \beta_i), \quad (1)$$

where k_i is the retrieval key, c_i is the memory content, and (α_i, β_i) are value parameters that define a Beta belief over memory value.

For each query, TTML first retrieves a Top- N candidate memory pool C_t from $\mathcal{M}_t$ and then uses a value score V_i computed from the value parameters to select Top- K memories A_t for the prompt. Conditioned on x_t and A_t , the frozen LLM samples reasoning trajectories:

$$\tau_{t,1:n} \sim p_\phi(\tau \mid x_t, A_t). \quad (2)$$

The trajectories yield a consensus reward R_t^{con} , which updates the value parameters of selected memories and guides memory expansion. Algorithm 1 gives the complete procedure; one TTML step is summarized as

$$\mathcal{M}_t \rightarrow C_t \rightarrow A_t \rightarrow \tau_{t,1:n} \rightarrow R_t^{\text{con}} \rightarrow \mathcal{M}_{t+1}. \quad (3)$$

3.2 Memory Policy Rollout

Value-based Memory Selection. With an embedding model f_{emb} , TTML first ranks memories by their relevance to the query,

$$\rho_{t,i} = \cos(f_{\text{emb}}(x_t), f_{\text{emb}}(k_i)), \quad (4)$$

and forms the candidate pool C_t by first filtering memories with $\rho_{t,i} \geq \delta$ and then keeping the Top- N among them. If no memory passes the gate, TTML executes the query without retrieved memories.

For each $m_i \in C_t$, TTML computes a value score V_i :

$$V_i^{\text{train}} \sim \text{Beta}(\alpha_i, \beta_i), \quad V_i^{\text{eval}} = \frac{\alpha_i}{\alpha_i + \beta_i}. \quad (5)$$

Following the two-stage test-time protocol in Section 4.1, TTML samples V_i^{train} during streaming memory accumulation to encourage exploration in memory selection, and uses the posterior mean

V_i^{eval} after the memory bank is fixed for evaluation. TTML then combines semantic relevance and memory value:

$$\text{score}_{t,i} = (1 - \lambda)z(\rho_{t,i}) + \lambda z(V_i), \quad (6)$$

where V_i denotes V_i^{train} during memory accumulation and V_i^{eval} during evaluation. Here, $z(\cdot)$ denotes z-score normalization within C_t , and $\lambda \in [0, 1]$ controls the trade-off. The Top- K memories are selected as

$$A_t = \text{TopK}_{m_i \in C_t}(\text{score}_{t,i}). \quad (7)$$

Memory-augmented Execution. TTML injects A_t into the executor prompt and samples n reasoning trajectories,

$$\tau_{t,1}, \dots, \tau_{t,n} \sim p_\phi(\tau \mid x_t, A_t). \quad (8)$$

The trajectories provide candidate answers for the current query and the evidence used to update memory values.

3.3 Bayesian Value Update

Consensus Reward R_t^{con} . Each trajectory $\tau_{t,j}$ is parsed into an answer $\hat{a}_{t,j}$. The majority answer is

$$\hat{a}_t = \text{mode}(\hat{a}_{t,1:n}). \quad (9)$$

The consensus reward is defined as the frequency of the majority answer among sampled trajectories:

$$R_t^{\text{con}} = \frac{1}{n} \sum_{j=1}^n \mathbb{1}[\hat{a}_{t,j} = \hat{a}_t]. \quad (10)$$

Bayesian Value Update. Since a prompt may contain multiple selected memories, the consensus reward R_t^{con} is observed only at the prompt level and cannot be directly attributed to an individual memory. TTML therefore assigns soft credit to each selected memory based on semantic relevance:

$$w_{t,i} = \frac{\exp\left(\frac{\rho_{t,i}}{\tau_c}\right)}{\sum_{m_\ell \in A_t} \exp\left(\frac{\rho_{t,\ell}}{\tau_c}\right)}, \quad m_i \in A_t, \quad (11)$$

where a smaller τ_c concentrates credit on the most relevant selected memories.

TTML then updates the value parameters with fractional evidence:

$$\begin{aligned} \alpha_i &\leftarrow \alpha_i + w_{t,i} R_t^{\text{con}}, \\ \beta_i &\leftarrow \beta_i + w_{t,i} (1 - R_t^{\text{con}}). \end{aligned} \quad (12)$$

High consensus adds positive evidence, while low consensus adds negative evidence. The update is local to memories in A_t , and the influence of a single noisy reward decreases as $\alpha_i + \beta_i$ grows.

3.4 Memory Bank Expansion

TTML expands the memory bank by extracting experience from the majority trajectory set:

$$\mathcal{T}_t^+ = \{\tau_{t,j} \mid \hat{a}_{t,j} = \hat{a}_t\}. \quad (13)$$

It then randomly samples one trajectory from this set:

$$\tau_t^* \sim \text{Uniform}(\mathcal{T}_t^+). \quad (14)$$

Under an extraction prompt, the frozen LLM acts as an Extractor and converts τ_t^* into a new memory candidate

$$\tilde{m}_t = (\tilde{k}_t, \tilde{c}_t), \quad (15)$$

After constructing the candidate, TTML expands the memory bank through either merging or insertion. If the maximum similarity between $\tilde{k}_t$ and existing keys exceeds the merge threshold η_{merge} , TTML merges the candidate with the most similar memory. Let

$$i^* = \arg \max_i \cos(f_{\text{emb}}(\tilde{k}_t), f_{\text{emb}}(k_i)). \quad (16)$$

The merge updates three parts of the memory. First, TTML replaces the old retrieval key with the new one, $k_{i^*} \leftarrow \tilde{k}_t$, so the memory reflects the latest trigger pattern. Second, TTML updates the content by keeping the new content $\tilde{c}_t$ and adding only non-redundant parts from the old content c_{i^*} . Concretely, both contents are split into sentences or bullet items; after punctuation is removed, an old item is discarded if its token overlap with any new item is above a de-duplication threshold. For example, if the new memory says “use parity to eliminate odd cases” and the old memory contains “check parity to rule out odd cases,” the old item is treated as redundant and removed. Third, the merged memory keeps the previous value parameters $(\alpha_{i^*}, \beta_{i^*})$, preserving the utility evidence accumulated before the merge. Otherwise, TTML inserts the candidate:

$$\tilde{m}_t = (\tilde{k}_t, \tilde{c}_t, \alpha_0, \beta_0), \quad \alpha_0 = \beta_0 = 1, \quad (17)$$

as a new memory with neutral value parameters, whose value is updated only when future queries select it and produce consensus rewards.

4 Experiments

4.1 Setup

Baselines. We compare TTML with three groups of label-free baselines. (1) Current-query optimization methods select or refine the answer for the current query only, including Self-Consistency (Wang

et al., 2023), Self-Certainty (Kang et al., 2025), and EM-INF (Agarwal et al., 2025). (2) Parameter-update methods are represented by TTRL, which updates model parameters from majority-vote rewards on unlabeled test data (Zuo et al., 2025). (3) Memory-accumulation methods are represented by ReasoningBank, which stores reusable reasoning strategies but does not explicitly manage memory or optimize memory selection (Ouyang et al., 2026). For the API-only DeepSeek-V4-Flash backbone, we omit Self-Certainty, EM-INF, and TTRL because the first two require full vocabulary distributions, while TTRL requires parameter updates.

Benchmarks. We evaluate TTML on six reasoning benchmarks. The mathematical suite includes AIME 2024, AIME 2025, AIME 2026, and MinervaMath, spanning contest-style and broader quantitative problems (Zhang and Math-AI, 2024, 2025, 2026; Lewkowycz et al., 2022). We use GPQA Diamond (GPQA) for expert-level scientific reasoning (Rein et al., 2023), and the Computer Science (CS) and Philosophy subsets of MMLU-Pro for disciplinary reasoning under a more challenging multi-choice format (Wang et al., 2024).

Backbone. We evaluate TTML with three backbones: Qwen3-8B (Qwen Team, 2025), Gemma-4-E4B-IT (Google DeepMind, 2026), and DeepSeek-V4-Flash (DeepSeek-AI, 2026). This model suite covers different model families and scales, allowing us to test whether TTML is robust across backbones rather than tied to a single generator.

Test-Time Protocol. Memory-based methods use a two-stage protocol. The memory bank is initialized as empty. In the test-time training stage, unlabeled queries arrive sequentially, and the memory bank is not reset between queries. This streaming setup allows experience extracted from earlier trajectories to be immediately integrated into the memory bank and used for subsequent queries. After this memory accumulation stage, we fix the accumulated memory bank and use it directly for retrieval and generation. ReasoningBank follows the same two-stage memory protocol, while others follow their original test-time protocols. Implementation details and prompt templates are provided in Appendices A and D.

4.2 Main Results

Table 1 shows that TTML achieves the highest Avg@8 for each of the three backbones across

Method	AIME 2024	AIME 2025	AIME 2026	GPQA	MinervaMath	CS	Philosophy	Average
Qwen3-8B								
Self-consistency	65.83	52.08	56.67	58.46	40.85	77.44	54.53	57.98
Self-certainty vote	67.50	51.67	56.67	58.27	38.24	79.21	57.11	58.38
EM-Inf	67.08	52.50	53.75	45.77	40.62	79.63	59.14	56.93
ReasoningBank	66.25	56.25	62.50	58.02	42.14	77.74	53.71	59.52
TTRL-best	67.92	53.75	57.50	56.31	44.72	78.05	56.54	59.26
No Memory	62.50	48.75	54.17	53.98	45.45	76.71	53.11	56.38
TTML	75.83	57.08	64.17	59.09	48.35	78.96	55.59	62.72
Gemma-4-E4B-IT								
Self-consistency	50.00	41.25	44.17	54.42	39.52	77.84	55.81	51.86
Self-certainty vote	50.42	41.67	45.83	54.61	35.66	78.17	55.84	51.74
EM-Inf	52.50	37.08	52.92	54.17	27.99	78.72	59.29	51.81
ReasoningBank	61.67	42.08	53.33	58.08	41.31	79.57	57.52	56.22
TTRL-best	57.08	42.50	55.00	55.81	40.76	78.84	57.77	55.39
No Memory	59.17	43.33	52.50	56.63	39.84	78.84	57.04	55.34
TTML	62.92	51.25	58.33	59.34	41.45	79.91	57.57	58.68
DeepSeek-V4-Flash								
Self-consistency	77.08	74.17	73.75	76.33	43.61	88.20	78.81	73.14
ReasoningBank	81.67	60.42	62.50	72.10	41.82	85.15	75.55	68.46
No Memory	82.50	71.25	73.75	78.03	44.62	87.80	77.81	73.68
TTML	90.00	76.67	79.17	80.24	46.65	89.09	79.73	77.36

Table 1: Avg@8 results compared with baselines across three backbone models and six benchmarks. **Bold** indicates the best result for each benchmark under each backbone.

six reasoning benchmarks. Compared with the strongest non-TTML baseline on each backbone, TTML improves the average score by 3.20, 2.46, and 3.68 points on Qwen3-8B, Gemma-4-E4B-IT, and DeepSeek-V4-Flash, respectively. The baseline comparisons clarify where the gains come from. On Qwen3-8B and Gemma-4-E4B-IT, the strongest current-query baselines reach 58.38 and 51.86 Avg@8, respectively, while TTML reaches 62.72 and 58.68. TTRL also remains below TTML, with 59.26 on Qwen3-8B and 55.39 on Gemma-4-E4B-IT, suggesting that writing noisy label-free feedback into parameters is not a reliable substitute for memory learning. ReasoningBank is the closest competitor on both models, with 59.52 on Qwen3-8B and 56.22 on Gemma-4-E4B-IT, but TTML still improves over it, showing that storing reusable experience is insufficient without value-based memory selection. Together, the results point to TTML’s key advantage: consensus feedback is accumulated as memory value and used to select useful memories for future queries, enabling persistent test-time learning without parameter updates.

4.3 Ablation Studies

Core Component Ablations. Table 2 removes three core mechanisms from TTML. When memory value is removed (w/o Memory Value), se-

lection relies only on semantic relevance; the average score drops by 2.36 points on Qwen3-8B and 1.93 points on Gemma-4-E4B-IT, showing that relevance alone cannot identify memories that have been reliable in past rollouts. When semantic relevance is removed (w/o Semantic Relevance), TTML selects memories by value without ensuring that they match the current query, leading to the largest average drop on both backbones. This shows that a high-value memory is useful only when it is relevant to the current query. Finally, removing memory merging (w/o Memory Merging) weakens memory consolidation. Without merging similar memories, redundant memories may fill the Top- K set, reducing memory diversity and preventing other useful memories from being selected. Overall, TTML works best when semantic relevance, learned memory value, and memory consolidation are combined.

Sensitivity to Top- K Memory Selection. Table 3 shows that selecting more memories is not always better. TTML achieves the best average score at $K = 3$ (65.69). Setting $K = 1$ reduces the average to 63.47 (−2.22 points), indicating the selected context is too narrow. Increasing to $K = 5$ also underperforms $K = 3$, at 63.61 (−2.08 points), suggesting larger memory sets introduce less rel-

Method	Qwen3-8B					Gemma-4-E4B-IT				
	AIME24	AIME25	AIME26	GPQA	Avg.	AIME24	AIME25	AIME26	GPQA	Avg.
TTML	75.83	57.08	64.17	59.09	64.04	62.92	51.25	58.33	59.34	57.96
w/o Memory Value	73.33	52.92	62.08	58.40	61.68	61.67	47.92	57.50	57.01	56.03
w/o Semantic Relevance	71.67	55.42	57.08	56.38	60.14	60.83	47.08	55.00	57.83	55.19
w/o Memory Merging	73.75	55.83	61.25	57.45	62.07	61.67	46.67	56.25	57.58	55.54

Table 2: Ablation study of TTML components across two backbone models.

Method	AIME24	AIME25	AIME26	Average
TTML ($K = 3$)	75.83	57.08	64.17	65.69
$K = 1$	74.58	56.25	59.58	63.47
$K = 5$	74.17	56.67	60.00	63.61

Table 3: Top- K memory sensitivity on Qwen3-8B.

Method	AIME24	AIME25	AIME26	Average
TTML ($\delta = 0.3$)	75.83	57.08	64.17	65.69
$\delta = 0.0$	67.50	55.00	60.83	61.11
$\delta = 0.1$	66.67	56.25	61.67	61.53
$\delta = 0.5$	74.58	55.42	59.58	63.19

Table 4: Sensitivity to retrieval threshold on Qwen3-8B.

evant memories. The best result at $K = 3$ shows that TTML benefits from selecting a small set of high-value memories that are relevant to the query and reliable under past consensus rewards, rather than adding more memories to the prompt.

Sensitivity to Retrieval Threshold. Table 4 evaluates the similarity threshold δ for candidate retrieval. The best average score is obtained at $\delta = 0.3$ (65.69). Lower thresholds substantially reduce performance, with averages of 61.11 at $\delta = 0.0$ and 61.53 at $\delta = 0.1$, indicating that value-based selection is vulnerable when the candidate pool contains many irrelevant memories. A stricter threshold $\delta = 0.5$ performs better (63.19) but still trails $\delta = 0.3$ by 2.50 points, suggesting that overly aggressive filtering can remove useful experience. Thus, TTML benefits from a moderately selective retrieval threshold: semantic filtering first forms a plausible candidate pool, and memory value then identifies the most useful memories within it.

4.4 Memory Transferability

Cross-backbone Transfer. Table 5 evaluates whether memories learned with one backbone can benefit another. In both transfer directions, TTML achieves the best performance on all evaluated benchmarks. When Qwen3-8B memories are used by Gemma-4-E4B-IT, TTML improves over ReasoningBank on every task (avg. +3.74

Qwen3-8B $\rightarrow$ Gemma-4-E4B-IT				
Method	AIME24	AIME25	AIME26	GPQA
No Memory	59.17	43.33	52.50	56.63
ReasoningBank	63.75	45.42	56.25	59.47
TTML	67.50	51.67	60.83	59.85

Gemma-4-E4B-IT $\rightarrow$ Qwen3-8B				
Method	AIME24	AIME25	AIME26	GPQA
No Memory	62.50	48.75	54.17	53.98
ReasoningBank	64.17	50.00	56.25	57.95
TTML	66.25	58.33	59.17	58.71

Table 5: Cross-backbone memory transfer results.

Source Memory: AIME24					
Method	AIME25	AIME26	GPQA	CS	Philosophy
No Memory	48.75	54.17	53.98	76.71	53.11
ReasoningBank	51.67	57.50	56.69	78.60	53.53
TTML	55.83	60.42	57.95	79.09	54.71

Source Memory: GPQA					
Method	AIME24	AIME25	AIME26	CS	Philosophy
No Memory	62.50	48.75	54.17	76.71	53.11
ReasoningBank	60.42	52.50	57.08	75.70	52.15
TTML	67.92	56.67	60.42	78.84	55.16

Table 6: Cross-benchmark memory transfer using Qwen3-8B for memory accumulation and evaluation.

points); the same trend holds when Gemma-4-E4B-IT memories are transferred to Qwen3-8B (avg. +3.52 points). This suggests that TTML memories capture reusable reasoning experience rather than model-specific artifacts, while value-based selection remains useful across executors.

Cross-benchmark Transfer. Table 6 further tests whether learned memories transfer across benchmarks. Memories constructed from AIME24 improve performance on AIME25 and AIME26, and also transfer to GPQA, CS, and Philosophy, with an average gain of 2.00 points over ReasoningBank. Similarly, memories constructed from GPQA improve both mathematical and non-mathematical targets, with an average gain of 4.23 points. TTML outperforms No Memory and ReasoningBank in all transfer settings. These transfer gains suggest that TTML stores reusable rather

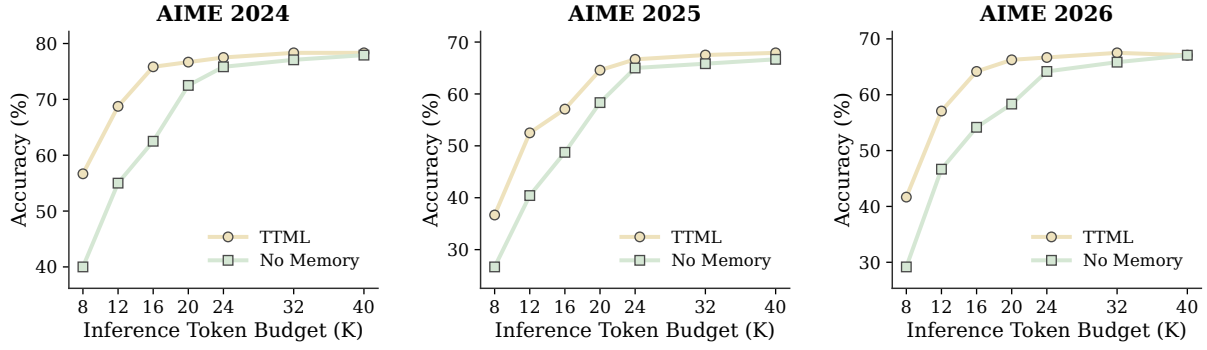


Figure 2: Accuracy across inference token budgets (i.e., Executor-generated tokens) on AIME with Qwen3-8B. TTML outperforms the no-memory baseline, especially at lower budgets.

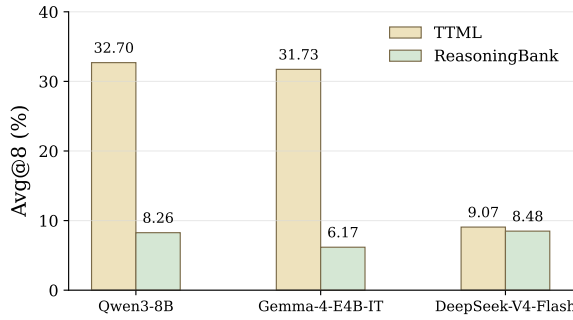


Figure 3: Avg@8 on no-memory-hard examples across all benchmarks.

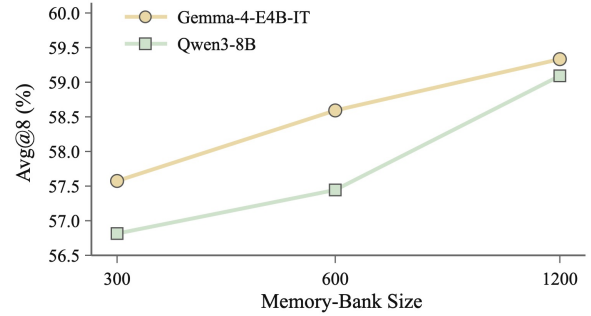


Figure 4: Avg@8 on GPQA across memory-bank sizes.

than benchmark-specific experience, and that value-based selection helps retrieve useful memories under shifted task distributions.

4.5 Inference Token Efficiency

Figure 2 evaluates accuracy under different inference token budgets on the AIME benchmarks. TTML consistently outperforms the No Memory baseline at the same budget, with the largest gaps in low-budget settings: at 8K tokens, average accuracy improves from 31.94% to 45.00%, and at 16K tokens from 55.14% to 65.69%. TTML with 16K tokens also exceeds No Memory with 20K tokens (65.69% vs. 63.06%), showing that memory can effectively reduce the budget needed to reach comparable accuracy. As the budget increases, the two curves gradually become closer, suggesting that larger budgets can partially compensate for the lack of memory through additional exploration. These results indicate that TTML improves token efficiency: by preserving reusable experience in memory, the model can reach strong accuracy with fewer inference tokens rather than relying on repeated trial-and-error generation.

4.6 Hard-Sample Recovery

We further evaluate TTML on no-memory-hard examples, where the No Memory baseline produces no correct answer. We report the unweighted average across benchmarks to measure whether memory recovers no-memory failures. Figure 3 shows that TTML substantially improves hard-sample recovery on Qwen3-8B and Gemma-4-E4B-IT, reaching 32.70% and 31.73% compared with 8.26% and 6.17% for ReasoningBank. On DeepSeek-V4-Flash, TTML is close to ReasoningBank (9.07% vs. 8.48%), suggesting that stronger backbones may leave fewer recoverable failures or use memory differently. Overall, TTML recovers some examples that are unsolved without memory, rather than only improving already solvable cases. Appendix E.1 provides a representative recovery case.

4.7 Memory-Bank Scaling

We further investigate whether TTML benefits from a larger memory bank. We evaluate Gemma-4-E4B-IT and Qwen3-8B on GPQA with 300, 600, and 1,200 memory entries, while fixing the number of retrieved memories at $K = 3$. As shown in Figure 4, increasing the bank size from 300 to

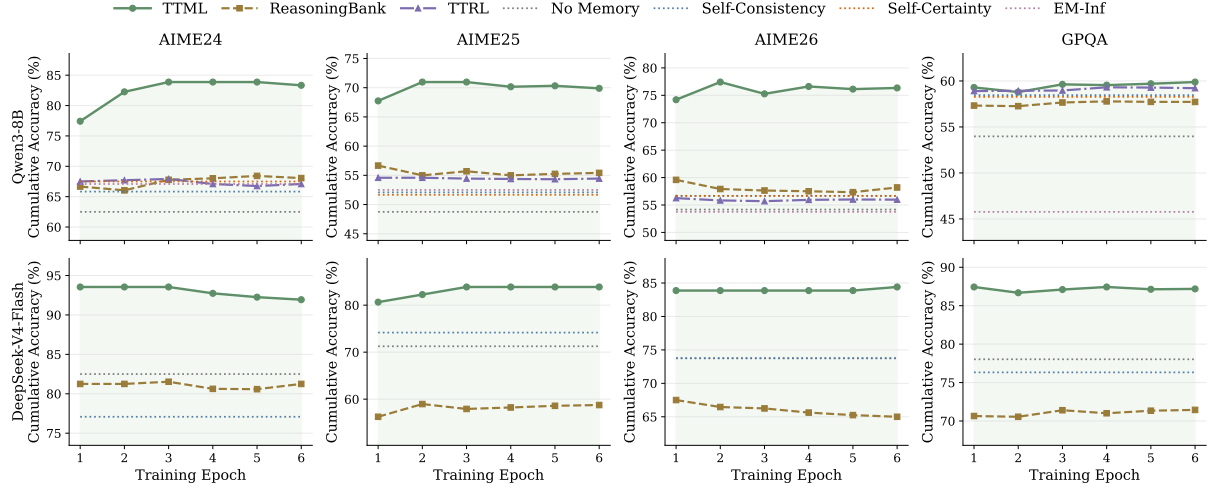


Figure 5: TTML accuracy trends across training epochs.

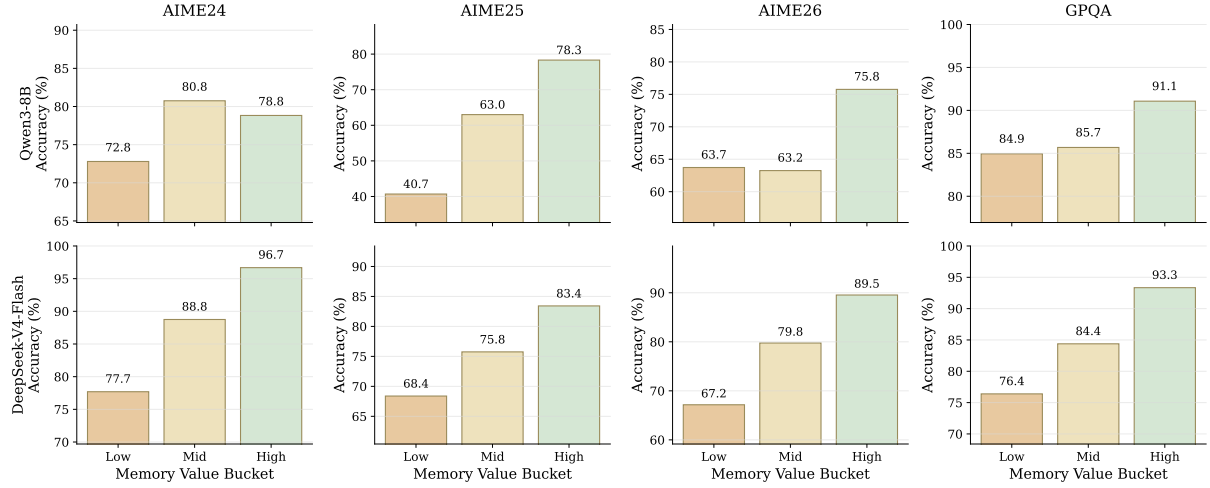


Figure 6: Accuracy under low-, mid-, and high-value memory groups, constructed by sorting memories by value score and splitting them into three equally sized buckets.

1,200 improves Avg@8 by 1.76 points on Gemma-4-E4B-IT and 2.27 points on Qwen3-8B. Since the retrieval budget is held fixed, the gains suggest that a larger memory bank improves the coverage and quality of retrievable experience rather than simply exposing the model to more context. This indicates that TTML can expand memory capacity without increasing per-query memory usage.

4.8 Cumulative Accuracy over Training

To evaluate TTML in streaming test-time training, we use cumulative accuracy—the ratio of correct predictions from epoch 1 to e . This metric mitigates per-epoch variance, capturing overall accumulated performance. Figure 5 shows that TTML generally improves or remains stable as more experience is accumulated. Its cumulative accuracy rises especially in the early epochs and stays com-

petitive or superior across later epochs, indicating that useful memories are quickly written and reused. Compared with ReasoningBank, TTML shows stronger and more stable accumulation behavior, while ReasoningBank often plateaus or declines as more self-generated memories are stored. These trends suggest that value-based memory updating and selection help TTML convert streaming trajectories into reusable experience rather than simply accumulating unfiltered memory entries. Appendix E.2 shows low-consensus trajectories yielding useful memories.

4.9 Memory Value Analysis

We further examine whether the learned memory value reflects the actual utility of memories. Specifically, we sort memories by their value scores and split them into three equally sized low-, mid-, and

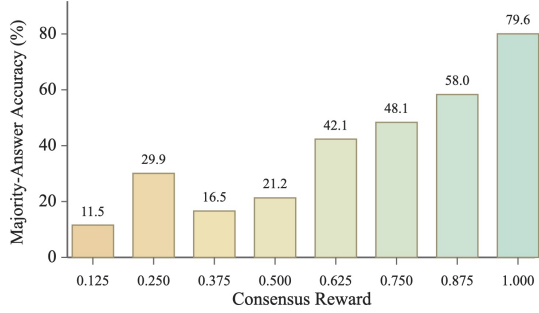


Figure 7: Majority accuracy across reward levels.

high-value buckets, and compare the resulting accuracy when memories from each bucket are used. Figure 6 shows that high-value memories consistently outperform low-value memories across both backbones and all evaluated benchmarks. The gap is especially clear on AIME25, where high-value memories improve accuracy over low-value memories from 40.7% to 78.3% on Qwen3-8B and from 68.4% to 83.4% on DeepSeek-V4-Flash. This indicates that TTML’s Bayesian value update learns a meaningful memory-selection signal: memories assigned higher values are more likely to support successful reasoning, which explains why value-based memory selection improves the final performance.

4.10 Reliability of Consensus Feedback

TTML uses R_{con} as label-free feedback for Bayesian value updates. We test whether stronger consensus indicates a more reliable majority answer by grouping predictions according to R_{con} and measuring majority-answer accuracy against ground truth. As shown in Figure 7, accuracy generally increases with consensus strength, from 11.5% at $R_{\text{con}} = 0.125$ to 79.6% under full consensus, despite small variations at low levels. These results show that R_{con} is not a correctness label, but a signal of answer reliability, making it suitable for Bayesian updates with feedback of varying strength. Importantly, even full consensus can be incorrect. This result further supports our design choice of applying consensus feedback to revisable external memory rather than directly to model parameters, allowing inaccurate feedback to be corrected by later evidence.

4.11 Memory Quality Analysis

We examine whether the memories produced by the Extractor contain valid and reusable experience. Qwen3.7-Max automatically evaluates each

Benchmark	Correctness	Non-redundancy	Generality
AIME24	93.46	81.31	98.13
AIME25	94.00	79.00	98.00
AIME26	88.52	89.34	95.90
GPQA	88.87	78.67	97.84

Table 7: Automatic evaluation (%) of extracted memories by Qwen3.7-Max using three binary criteria.

extracted memory using three binary criteria: correctness, whether the method is valid for the task; non-redundancy, whether the content avoids excessive repetition; and generality, whether the experience can be reused for similar problems. Table 7 shows that correctness ranges from 88.52% to 94.00%, while generality remains above 95.90% across all four benchmarks. Non-redundancy is lower, ranging from 78.67% to 89.34%, indicating that some memories still contain repeated information. Overall, the extracted memories are largely valid and reusable, while the lower non-redundancy scores highlight the importance of memory merging for controlling duplication.

4.12 Runtime Overhead

We further analyze the runtime overhead of TTML from two perspectives. First, we track runtime as the memory bank grows and find that retrieval and memory updates account for at most 1.72% of the total cost, with end-to-end runtime remaining stable across six epochs. Second, with eight roll-outs, TTML has comparable per-query runtime to Self-Consistency, Self-Certainty, and TTRL, while being substantially faster than EM-INF; Reasoning-Bank is cheaper, and TTRL additionally requires four GPUs. Overall, TTML supports persistent test-time learning with limited additional runtime overhead. Detailed results are in Appendix C.

5 Conclusion

We introduced TTML, a Test-Time Memory Learning framework for label-free RLVR. TTML shifts the optimization target from single-query outputs or model parameters to an external memory bank. By combining value-based memory selection, Bayesian value estimation, and memory expansion, TTML turns one-off reasoning trajectories into reusable experience that can benefit future unlabeled queries. Across six benchmarks and three backbones, TTML outperforms strong label-free baselines on average, highlighting memory as an effective optimization target at test time.

Limitations

TTML relies on consensus rewards derived from sampled answers, and is therefore most directly applicable to tasks where answer equivalence can be reliably parsed or normalized, such as mathematical reasoning and multiple-choice science questions. For more open-ended settings, including code generation or long-horizon interactive tasks, memory learning may require executable feedback, task-specific normalization, or external verifiers to define a reliable consensus signal. In addition, our soft credit assignment uses key-query similarity as a lightweight approximation of memory contribution. While this keeps test-time learning simple and local, future work could improve credit assignment with intervention-based memory dropout or finer-grained attribution. The effect of an isolated erroneous feedback signal is bounded, and later low-consensus feedback can reduce the value of the affected memory. However, this correction mechanism may be less effective when the model repeatedly produces the same incorrect answer with high consensus.

Acknowledgments

This work is supported by the National Science and Technology Major Project (Grant No. 2022ZD0116101), the Project of Research and Development for Neural Machine Translation Models between Cantonese and Mandarin (Grant No. WT135-76), the Major Scientific Research Project of the State Language Commission in the 13th Five-Year Plan (Grant No. WT135-38), and the Public Technology Service Platform Project of Xiamen City (Grant No. 3502Z20231043).

References

- Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han, and Hao Peng. 2025. [The unreasonable effectiveness of entropy minimization in llm reasoning](#). *Preprint*, arXiv:2505.15134.
- Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. [Large language monkeys: Scaling inference compute with repeated sampling](#). *Preprint*, arXiv:2407.21787.
- Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. [M3-embedding: Multilinguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation](#). *Preprint*, arXiv:2402.03216.
- Xinjie Chen, Biao Fu, Jing Wu, Guoxin Chen, Xinggao Liu, Dayiheng Liu, and Minpeng Liao. 2026a. [Jury-r1: Votes propose, proofs dispose for label-free rlvr](#). *Preprint*, arXiv:2604.25419.
- Xinjie Chen, Minpeng Liao, Guoxin Chen, Chengxi Li, Biao Fu, Kai Fan, and Xinggao Liu. 2026b. [From data-centric to sample-centric: Enhancing LLM reasoning via progressive optimization](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2227–2242, San Diego, California, United States. Association for Computational Linguistics.
- DeepSeek-AI. 2026. [DeepSeek-V4: Towards highly efficient million-token context intelligence](#). *Preprint*, arXiv:2606.19348.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shiron Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). *CoRR*, abs/2501.12948.
- Kaixuan Du, Meng Cao, Hang Zhang, Yukun Wang, Xiangzhou Huang, and Ni Li. 2026. [Dual consensus: Escaping from spurious majority in unsupervised rlvr via two-stage vote mechanism](#). *Preprint*, arXiv:2603.16223.
- Google DeepMind. 2026. [Gemma 4 E4B-IT](#). Hugging Face model card.
- Bingxiang He, Yuxin Zuo, Zeyuan Liu, Shangzhi Zhao, Zixuan Fu, Junlin Yang, Cheng Qian, Kaiyan Zhang, Yuchen Fan, Ganqu Cui, Xiusi Chen, Youbang Sun, Xingtai Lv, Xuekai Zhu, Li Sheng, Ran Li, Huan ang Gao, Yuchen Zhang, Bowen Zhou, and 2 others. 2026. [How far can unsupervised rlvr scale llm training?](#) *Preprint*, arXiv:2603.08660.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helvar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, and 242 others. 2024. [Openai o1 system card](#). *Preprint*, arXiv:2412.16720.
- Zhewei Kang, Xuandong Zhao, and Dawn Song. 2025. [Scalable best-of-n selection for large language models via self-certainty](#). *Preprint*, arXiv:2502.18581.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. [Solving quantitative reasoning problems with language models](#). *Preprint*, arXiv:2206.14858.

- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, 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](#). *Preprint*, arXiv:2303.17651.
- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. [Training language models to follow instructions with human feedback](#). *CoRR*, abs/2203.02155.
- Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2026. [Reasoning-Bank: Scaling agent self-evolving with reasoning memory](#). In *The Fourteenth International Conference on Learning Representations*.
- Teng Pan, Yuchen Yan, Zixuan Wang, Ruiqing Zhang, Guiyang Hou, Wenqi Zhang, Weiming Lu, Jun Xiao, and Yongliang Shen. 2026. [Coverrl: Breaking the consensus trap in label-free reasoning via generator-verifier co-evolution](#). *Preprint*, arXiv:2603.17775.
- Mihir Prabhudesai, Lili Chen, Alex Ippoliti, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. 2025. [Maximizing confidence alone improves reasoning](#). *CoRR*, abs/2505.22660.
- Archiki Prasad, Weizhe Yuan, Richard Yuanzhe Pang, Jing Xu, Maryam Fazel-Zarandi, Mohit Bansal, Sainbayar Sukhbaatar, Jason Weston, and Jane Yu. 2025. [Self-consistency preference optimization](#). *Preprint*, arXiv:2411.04109.
- Qwen Team. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dierani, Julian Michael, and Samuel R. Bowman. 2023. [Gpqa: A graduate-level google-proof q&a benchmark](#). *Preprint*, arXiv:2311.12022.
- Sheikh Shafayat, Fahim Tajwar, Ruslan Salakhutdinov, Jeff Schneider, and Andrea Zanette. 2025. [Can large reasoning models self-train?](#) *CoRR*, abs/2505.21444.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *CoRR*, abs/2402.03300.
- 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 *International Conference on Learning Representations*.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhui Chen. 2024. [Mmlu-pro: A more robust and challenging multi-task language understanding benchmark](#). *Preprint*, arXiv:2406.01574.
- Muyang Ye, Tian Lan, Feihu Jiang, Yongshi Ye, Wuyunsiqin, Bin Zhu, Qianghuai Jia, Zhao Xu, Weihua Luo, Ye Wang, Jinyang Zhang, Longyue Wang, and Lingfeng Bao. 2026a. [Search2skill: Skill distillation beyond knowledge boundaries via rubric-based reinforcement learning](#). *Preprint*, arXiv:2608.05245.
- Yongshi Ye, Hui Jiang, Feihu Jiang, Tian Lan, Yichao Du, Biao Fu, Xiaodong Shi, Qianghuai Jia, Longyue Wang, and Weihua Luo. 2026b. [Umem: Unified memory extraction and management framework for generalizable memory](#). *Preprint*, arXiv:2602.10652.
- Yongshi Ye, Liang Zhang, Yidong Chen, Xiaodong Shi, and Biao Fu. 2026c. [Don't peek at the answer: Outcome-masked group relative policy optimization for label-free rlvr](#). *Preprint*, arXiv:2608.03119.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. [Dapo: An open-source llm reinforcement learning system at scale](#). *Preprint*, arXiv:2503.14476.
- Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. [Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?](#) *Preprint*, arXiv:2504.13837.
- Qingyang Zhang, Haitao Wu, Changqing Zhang, Peilin Zhao, and Yatao Bian. 2025. [Right question is already half the answer: Fully unsupervised llm reasoning incentivization](#). *Preprint*, arXiv:2504.05812.
- Yifan Zhang and Team Math-AI. 2024. [American invitational mathematics examination \(AIME\) 2024](#). Hugging Face dataset.
- Yifan Zhang and Team Math-AI. 2025. [American invitational mathematics examination \(AIME\) 2025](#). Hugging Face dataset.
- Yifan Zhang and Team Math-AI. 2026. [American invitational mathematics examination \(AIME\) 2026](#). Hugging Face dataset.
- Zizhuo Zhang, Jianing Zhu, Xinmu Ge, Zihua Zhao, Zhanke Zhou, Xuan Li, Xiao Feng, Jiangchao Yao,

and Bo Han. 2026. [Co-rewarding: Stable self-supervised RL for eliciting reasoning in large language models](#). In *The Fourteenth International Conference on Learning Representations*.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. 2025. [Absolute zero: Reinforced self-play reasoning with zero data](#). *CoRR*, abs/2505.03335.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. 2025. [Ttrl: Test-time reinforcement learning](#). *Preprint*, arXiv:2504.16084.

A Implementation Details

Unless otherwise specified, all main experiments use the hyperparameters in Table 8. Ground-truth answers are never used for reward updates during test-time adaptation; they are used only for offline evaluation and logging. For baselines, parameters not reported in our tables follow the best task-specific configurations reported in the original papers. We use BGE-M3 (Chen et al., 2024) as the embedding model f_{emb} . The LLM parameters remain frozen throughout test-time adaptation; only the external memory bank is updated.

B Theoretical Analysis

This section analyzes the memory value learning mechanism in TTML. TTML keeps the LLM frozen and adapts an external memory bank by updating memory value parameters (α_i, β_i) with label-free consensus feedback.

B.1 Majority-Answer Frequency as Label-Free Feedback

Given a test-time query x_t and selected memories A_t , the frozen LLM samples n reasoning trajectories:

$$\tau_{t,1}, \dots, \tau_{t,n} \sim p_\phi(\tau \mid x_t, A_t). \quad (18)$$

Each trajectory $\tau_{t,j}$ is parsed into a candidate answer $\hat{a}_{t,j}$. The majority answer is

$$\hat{a}_t = \text{mode}(\hat{a}_{t,1:n}), \quad (19)$$

and the consensus reward is defined as

$$R_t^{\text{con}} = \frac{1}{n} \sum_{j=1}^n \mathbb{1}[\hat{a}_{t,j} = \hat{a}_t]. \quad (20)$$

Thus, $R_t^{\text{con}} \in [0, 1]$ measures how concentrated the sampled answers are under the current query-memory context.

This choice is motivated by self-consistency and repeated-sampling based test-time scaling methods, where aggregating multiple sampled reasoning paths often improves reasoning performance over single-path decoding (Wang et al., 2023; Brown et al., 2024). Recent label-free test-time learning methods also use agreement statistics from sampled outputs as feedback when ground-truth labels are unavailable (Zuo et al., 2025).

Consensus calibration. In answer-verifiable reasoning tasks, we use the following calibration condition: higher concentration in the sampled answer distribution is positively correlated with more reliable reasoning behavior. Under this condition, R_t^{con} can serve as a label-free proxy signal for updating the value posterior of the memories used in the prompt.

B.2 Statistical Interpretation of Empirical Answer Frequency

Consider a fixed query x with ground-truth answer y^* . Given a memory set A , let

$$q = p(y^* \mid x, A) \quad (21)$$

be the probability that one sampled trajectory produces the correct answer. For n independent samples, define

$$K = \sum_{j=1}^n \mathbb{1}[\hat{a}_j = y^*]. \quad (22)$$

Then

$$K \sim \text{Binomial}(n, q), \quad (23)$$

and the empirical frequency of the correct answer is

$$\hat{q} = \frac{K}{n}, \quad \mathbb{E}[\hat{q}] = q. \quad (24)$$

When the majority answer equals y^* , the consensus reward coincides with the empirical correct-answer frequency:

$$R^{\text{con}} = \hat{q}. \quad (25)$$

Now consider two memory states during memory accumulation. Suppose that after memory learning, the probability of sampling the correct answer increases from q_e to

$$q_{e+1} = q_e + \Delta, \quad \Delta > 0. \quad (26)$$

Then

$$\mathbb{E}[\hat{q}_{e+1}] - \mathbb{E}[\hat{q}_e] = \Delta. \quad (27)$$

For finite samples, the observed empirical frequency may fluctuate. By Hoeffding's inequality and a union bound,

$$\Pr(\hat{q}_{e+1} < \hat{q}_e) \leq 2 \exp\left(-\frac{n\Delta^2}{2}\right). \quad (28)$$

Therefore, when memory learning increases the sampling probability of the target answer, the empirical answer frequency is more likely to increase as the number of sampled trajectories grows. This provides a statistical motivation for using majority-answer frequency as cumulative feedback in TTML.

B.3 Memory Value and Beta Posterior Estimation

TTML represents each memory as

$$m_i = (k_i, c_i, \alpha_i, \beta_i), \quad (29)$$

where k_i is the retrieval key, c_i is the memory content, and (α_i, β_i) are value parameters. These parameters define a Beta belief over memory value:

$$\theta_i \sim \text{Beta}(\alpha_i, \beta_i). \quad (30)$$

We define the proxy value of memory m_i as

$$U_i^{\text{proxy}} = \mathbb{E}[R_t^{\text{con}} \mid m_i \text{ is used in similar queries}]. \quad (31)$$

This quantity measures the expected consensus level induced when m_i is selected for similar test-time queries.

We first consider the single-memory case $A_t = \{m_i\}$. Suppose m_i is used for T similar queries and receives feedback

$$R_1^{\text{con}}, R_2^{\text{con}}, \dots, R_T^{\text{con}}, \quad R_t^{\text{con}} \in [0, 1]. \quad (32)$$

TTML updates the value parameters with fractional evidence:

$$\alpha_i^{(T)} = \alpha_0 + \sum_{t=1}^T R_t^{\text{con}}, \quad (33)$$

$$\beta_i^{(T)} = \beta_0 + \sum_{t=1}^T (1 - R_t^{\text{con}}). \quad (34)$$

The posterior mean used during evaluation is

$$V_i^{\text{eval},(T)} = \frac{\alpha_i^{(T)}}{\alpha_i^{(T)} + \beta_i^{(T)}}. \quad (35)$$

Substituting the updates gives

$$V_i^{\text{eval},(T)} = \frac{\alpha_0 + \sum_{t=1}^T R_t^{\text{con}}}{\alpha_0 + \beta_0 + T}. \quad (36)$$

Under a locally stationary distribution of similar queries, assume

$$\mathbb{E}[R_t^{\text{con}} \mid m_i \text{ is used}] = U_i^{\text{proxy}}. \quad (37)$$

By the law of large numbers,

$$\frac{1}{T} \sum_{t=1}^T R_t^{\text{con}} \rightarrow U_i^{\text{proxy}}. \quad (38)$$

Therefore,

$$V_i^{\text{eval},(T)} \rightarrow U_i^{\text{proxy}}, \quad \text{as } T \rightarrow \infty. \quad (39)$$

This shows that the Beta posterior mean estimates the expected label-free proxy value of a memory under repeated use in similar contexts.

B.4 Soft Credit Assignment for Multi-Memory Prompts

In TTML, a prompt typically contains multiple selected memories:

$$A_t = \{m_{i_1}, m_{i_2}, \dots, m_{i_K}\}. \quad (40)$$

The consensus reward R_t^{con} is observed at the prompt level. TTML distributes this feedback to selected memories according to query-memory relevance.

For each selected memory $m_i \in A_t$, the semantic relevance is

$$\rho_{t,i} = \cos(f_{\text{emb}}(x_t), f_{\text{emb}}(k_i)). \quad (41)$$

TTML converts these relevance scores into soft credit weights:

$$w_{t,i} = \frac{\exp(\rho_{t,i}/\tau_c)}{\sum_{m_\ell \in A_t} \exp(\rho_{t,\ell}/\tau_c)}, \quad m_i \in A_t, \quad (42)$$

where τ_c is the credit assignment temperature. A smaller τ_c concentrates credit on the most relevant selected memories, while a larger τ_c yields a smoother credit distribution.

Since

$$\sum_{m_i \in A_t} w_{t,i} = 1, \quad (43)$$

each prompt-level feedback contributes one unit of fractional evidence across the selected memories. For memory m_i , the positive evidence is

$$w_{t,i} R_t^{\text{con}}, \quad (44)$$

and the negative evidence is

$$w_{t,i} (1 - R_t^{\text{con}}). \quad (45)$$

The Bayesian value update is therefore

$$\begin{aligned} \alpha_i &\leftarrow \alpha_i + w_{t,i} R_t^{\text{con}}, \\ \beta_i &\leftarrow \beta_i + w_{t,i} (1 - R_t^{\text{con}}), \end{aligned} \quad m_i \in A_t. \quad (46)$$

For memories not selected into the prompt, $m_i \notin A_t$, the value parameters remain unchanged. This update makes memory value learning local to selected memories and aligns with the Bayesian value update in Section 3.3.

B.5 Bounded Influence of One-Step Updates

We next show that the effect of a single consensus reward on the posterior mean is bounded and decreases as evidence accumulates.

Let the current posterior mean of memory m_i be

$$\mu_i = V_i^{\text{eval}} = \frac{\alpha_i}{\alpha_i + \beta_i}, \quad (47)$$

and define the evidence strength

$$S_i = \alpha_i + \beta_i. \quad (48)$$

For one new prompt-level feedback, write

$$R = R_t^{\text{con}}, \quad w = w_{t,i}, \quad (49)$$

where $R \in [0, 1]$ and $w \in [0, 1]$. After the update, the posterior mean becomes

$$\mu'_i = \frac{\alpha_i + wR}{S_i + w}. \quad (50)$$

Then

$$\mu'_i - \mu_i = \frac{\alpha_i + wR}{S_i + w} - \frac{\alpha_i}{S_i} \quad (51)$$

$$= \frac{S_i(\alpha_i + wR) - \alpha_i(S_i + w)}{S_i(S_i + w)} \quad (52)$$

$$= \frac{w(S_i R - \alpha_i)}{S_i(S_i + w)} \quad (53)$$

$$= \frac{w(R - \mu_i)}{S_i + w}. \quad (54)$$

Since $R \in [0, 1]$ and $\mu_i \in [0, 1]$,

$$|R - \mu_i| \leq 1. \quad (55)$$

Thus,

$$|\mu'_i - \mu_i| \leq \frac{w}{S_i + w}. \quad (56)$$

Because $w \leq 1$, we also have

$$|\mu'_i - \mu_i| \leq \frac{1}{S_i + 1}. \quad (57)$$

This bound shows that new memories with small S_i can adapt quickly, while memories with larger accumulated evidence are less sensitive to a single noisy reward.

B.6 Thompson Sampling for Test-Time Exploration

During streaming memory accumulation, TTML uses Thompson sampling to compute the memory value score:

$$V_i^{\text{train}} \sim \text{Beta}(\alpha_i, \beta_i), \quad m_i \in C_t. \quad (58)$$

After the memory bank is fixed for evaluation, TTML uses the posterior mean:

$$V_i^{\text{eval}} = \frac{\alpha_i}{\alpha_i + \beta_i}. \quad (59)$$

The variance of the Beta posterior is

$$\text{Var}[\theta_i | \alpha_i, \beta_i] = \frac{\alpha_i \beta_i}{(\alpha_i + \beta_i)^2 (\alpha_i + \beta_i + 1)}. \quad (60)$$

Therefore, memories with limited evidence have higher posterior uncertainty and may occasionally sample high values, allowing them to be explored. Memories that repeatedly receive high consensus rewards concentrate their posterior mass in a high-value region and are selected more consistently. Memories with repeatedly low consensus rewards concentrate toward lower values and are selected less often.

The sampled or deterministic value score is combined with semantic relevance as

$$\text{score}_{t,i} = (1 - \lambda)z(\rho_{t,i}) + \lambda z(V_i), \quad (61)$$

where $V_i = V_i^{\text{train}}$ during streaming memory accumulation and $V_i = V_i^{\text{eval}}$ during evaluation. Here, $z(\cdot)$ denotes z-score normalization within C_t , and λ controls the trade-off between semantic relevance and memory value. Thus, Thompson sampling provides posterior-driven exploration during memory accumulation, while posterior-mean selection provides deterministic behavior during evaluation.

C Computational Cost Analysis

We report two complementary cost profiles. First, we profile TTML over six GPQA epochs to examine how its internal cost changes as the memory bank grows. Table 9 shows that Executor generation dominates runtime, accounting for 90.41% on Gemma-4-E4B-IT and 93.27% on Qwen3-8B. Retrieval and memory update together account for only 1.72% and 0.89%, respectively. Although memory-update time increases with the bank size, the total per-epoch runtime remains within a stable range across the six epochs.

Hyperparameter	Symbol	Value	Description
Similarity gate	δ	0.30	Minimum key-query similarity for candidate memories.
Candidate pool size	N	10	Number of retrieved memories in C_t .
Injected memories	K	3	Number of selected memories in A_t .
Relevance-value weight	λ	0.50	Weight for combining $z(\rho_{t,i})$ and $z(V_i)$.
Credit temperature	τ_c	0.10	Temperature for soft credit assignment.
Memory prior	α_0, β_0	1.0, 1.0	Initial value parameters for inserted memories.
Merge threshold	η_{merge}	0.90	Key-similarity threshold for merging a candidate memory.
Trajectory samples	n	8	Number of reasoning trajectories sampled per query.
Executor temperature	–	0.60	Sampling temperature for self-consistency rollouts.
Executor top- p	–	1.00	Nucleus sampling threshold.
Executor max tokens	–	16384	Maximum generation length for rollouts.
Extractor temperature	–	0.00	Deterministic memory extraction.
Extractor top- p	–	1.00	Nucleus sampling threshold.
Extractor max tokens	–	4096	Maximum generation length for memory extraction.
Batch size	–	AIME series: 30; others: 32	Fixed by task type.
Reported epoch	–	6	Epoch used for the main TTML results.

Table 8: Implementation hyperparameters used in the main experiments.

Second, we compare end-to-end cost across methods under the same eight-rollout setting on AIME24 with Gemma-4-E4B-IT. As shown in Table 10, TTML takes 86.47 seconds and generates 34,173 tokens per query. Its runtime is comparable to Self-Consistency (93.10 seconds) and Self-Certainty (91.33 seconds), and substantially lower than EM-INF (852.04 seconds), while Reasoning-Bank is cheaper at 55.68 seconds. TTRL takes 90.67 seconds but uses four GPUs. Within TTML, the Extractor adds 2.56 seconds and 5,558 tokens, whereas retrieval and memory update together take less than one second. Thus, most of TTML’s cost comes from the eight Executor rollouts rather than memory maintenance.

D Prompt Templates

We use the same frozen LLM under two prompting roles: the Executor solves the current query (Prompts 1 and 2), while the Extractor converts the selected trajectory into a reusable memory (Prompts 3 and 4). The No Memory baseline uses the corresponding executor template with `{retrieved_memories}` set to (none). For memory-augmented execution, `{retrieved_memories}` contains only the experience field of each selected memory, rather than the full memory schema.

E Case Studies

E.1 Memory helps recover no-memory failures.

Figure 8 shows a no-memory-hard AIME example where all eight no-memory samples fail and con-

verge to an incorrect majority answer. The failure comes from a local construction trap: the model fixes an equilateral substructure early, derives vertices that satisfy local equal-distance constraints, but violates the global constraint $BC = 38$. TTML retrieves a reusable coordinate-geometry memory that encourages keeping global constraints active and computing the area symbolically. With this memory, the model preserves the hypotenuse constraint throughout the derivation and recovers the correct majority answer. This case illustrates the motivation of TTML: test-time trajectories can be distilled into reusable experience that helps later queries avoid repeated reasoning failures without updating model parameters.

E.2 Low-consensus trajectories can still yield useful memory.

Figure 9 shows a training-process AIME example where an early epoch has weak consensus ($R_{\text{con}} = 0.125$), so the rollout does not provide a reliable answer. TTML nevertheless extracts a reusable structural cue: the constraint on possible square-root sums should trigger sign-symmetry reasoning and Vieta’s formulas. The resulting memory is not an answer cache, but a reusable reasoning pattern. After memory accumulation, the same cue guides the executor to a stable solution, with all final samples producing the correct answer. This case illustrates TTML’s motivation: imperfect test-time trajectories can still be converted into reusable memory that improves later reasoning without updating model parameters.

Algorithm 1: Test-Time Memory Learning (TTML)

Input: Unlabeled test stream $\mathcal{D}_u = \{x_t\}_{t=1}^T$; initial memory bank $\mathcal{M}_1 = \emptyset$; frozen LLM p_ϕ ; embedding model f_{emb}

Output: Predictions $\{\hat{a}_t\}_{t=1}^T$ and updated memory bank $\mathcal{M}_{T+1}$

```
1 for  $t = 1, \dots, T$  do
2   Compute  $\rho_{t,i} = \cos(f_{\text{emb}}(x_t), f_{\text{emb}}(k_i))$  for each  $m_i \in \mathcal{M}_t$ ;
3   Retrieve the Top- $N$  candidate memory pool  $C_t$ ;
4   if  $C_t = \emptyset$  then
5     Set  $A_t = \emptyset$ ;
6   else
7     foreach  $m_i \in C_t$  do
8       Set  $V_i \sim \text{Beta}(\alpha_i, \beta_i)$  for online adaptation;
          // Use  $V_i = \alpha_i / (\alpha_i + \beta_i)$  for deterministic evaluation
9     Compute  $\text{score}_{t,i} = (1 - \lambda)z(\rho_{t,i}) + \lambda z(V_i)$  for each  $m_i \in C_t$ ;
10    Select  $A_t = \text{TopK}_{m_i \in C_t}(\text{score}_{t,i})$ ;
11    Sample trajectories  $\tau_{t,1:n} \sim p_\phi(\tau \mid x_t, A_t)$ ;
12    Parse answers  $\hat{a}_{t,1:n}$  and set  $\hat{a}_t = \text{mode}(\hat{a}_{t,1:n})$ ;
13    Compute  $R_t^{\text{con}} = \frac{1}{n} \sum_{j=1}^n \mathbb{1}[\hat{a}_{t,j} = \hat{a}_t]$ ;
14    foreach  $m_i \in A_t$  do
15      Compute  $w_{t,i} = \frac{\exp(\frac{\rho_{t,i}}{\tau_c})}{\sum_{m_\ell \in A_t} \exp(\frac{\rho_{t,\ell}}{\tau_c})}$ ;
16      Update  $\alpha_i \leftarrow \alpha_i + w_{t,i} R_t^{\text{con}}, \quad \beta_i \leftarrow \beta_i + w_{t,i} (1 - R_t^{\text{con}})$ ;
17    Form  $\mathcal{T}_t^+ = \{\tau_{t,j} \mid \hat{a}_{t,j} = \hat{a}_t\}$  and sample  $\tau_t^* \sim \text{Uniform}(\mathcal{T}_t^+)$ ;
18    Extract a memory candidate  $(\tilde{k}_t, \tilde{c}_t)$  from  $\tau_t^*$ ;
19    if  $\max_{m_i \in \mathcal{M}_t} \cos(f_{\text{emb}}(\tilde{k}_t), f_{\text{emb}}(k_i)) \geq \eta_{\text{merge}}$  then
20      Merge the candidate with the memory of highest key similarity;
21    else
22      Insert  $\tilde{m}_t = (\tilde{k}_t, \tilde{c}_t, \alpha_0, \beta_0)$  into  $\mathcal{M}_t$ ;
23    Set  $\mathcal{M}_{t+1}$  to the updated memory bank;
```

Module	Gemma-4-E4B-IT		Qwen3-8B	
	Time (s)	Share	Time (s)	Share
Retrieval	1.8–2.1	0.24%	1.9–2.2	0.13%
Executor	722.1–836.6	90.41%	1481.6–1569.9	93.27%
Extractor	58.7–64.3	7.32%	80.0–115.7	5.56%
Memory update	8.7–15.1	1.48%	8.6–15.1	0.76%
Other	4.4–5.0	0.55%	4.4–4.8	0.28%
Total	807.9–915.3	100%	1581.8–1705.0	100%

Table 9: End-to-end TTML runtime over six GPQA epochs. Time is the minimum–maximum per-epoch runtime, and Share is computed from cumulative runtime over all six epochs.

Method	Phase	Time (s)	Share	Tokens
Self-Consistency	Rollout generation	93.08	99.98%	31,155
	Other local operations	0.02	0.02%	0
	Total (1 GPU)	93.10	100%	31,155
Self-Certainty	Rollout generation	89.17	97.63%	31,155
	Confidence scoring	2.16	2.37%	0
	Total (1 GPU)	91.33	100%	31,155
EM-INF	Setup/load	3.03	0.36%	0
	Rollout generation	818.66	96.08%	26,814
	EM processor	30.31	3.56%	0
	Other local operations	0.04	0.00%	0
	Total (1 GPU)	852.04	100%	26,814
ReasoningBank	Executor generation	51.25	92.04%	2,730
	Extractor generation	4.19	7.53%	2,022
	Retrieval/memory update	0.21	0.38%	0
	Other local operations	0.03	0.05%	0
	Total (1 GPU)	55.68	100%	4,752
TTRL	Rollout generation	57.42	63.33%	23,100
	Log-probability/reward/advantage	11.25	12.41%	–
	Actor update	13.88	15.31%	–
	Weight synchronization	8.08	8.91%	0
	Other operations	0.04	0.04%	0
	Total (4 GPUs)	90.67	100%	23,100
TTML	Executor generation	83.25	96.27%	28,615
	Extractor generation	2.56	2.96%	5,558
	Memory update	0.64	0.75%	0
	Retrieval/other	0.02	0.02%	0
	Total (1 GPU)	86.47	100%	34,173

Table 10: Per-query end-to-end cost on AIME24 with Gemma-4-E4B-IT. All methods use eight rollouts. TTRL uses four H100 80GB GPUs, while all other methods use one. Tokens serve as a generation-cost proxy; “–” denotes non-generation operations for which token counts do not apply.

```

# Role
You are an expert reasoning assistant.

# Goal
Solve the given problem.

# Retrieved Memories (may help)
{retrieved_memories}

# Output Format
- First, provide your reasoning.
- Then, provide your final answer strictly in:
  <answer>...</answer>

---

# Problem
{problem}

```

Executor prompt for free-form answer tasks.

```

# Role
You are an expert multiple-choice question solver.

# Goal
Select the single best option.

# Retrieved Memories (may help)
{retrieved_memories}

# Output Format
- Provide reasoning.
- Then output the option label strictly in:
  <answer>...</answer>

# Important
- Output ONLY the option label (e.g., A, F, or J) inside <answer>.

---

# Question
You are given a multiple-choice question. Select the best option.

Question:
{question}

Options:
{label_1}. {choice_1}
{label_2}. {choice_2}
...

Answer with the option label (e.g., A, F, or J).

```

Executor prompt for multiple-choice tasks.

```

# Role
You are a Memory Extractor model. You observe a model's solution attempt and extract one reusable
experience item.

# Context
Problem:
{problem}

Model Solution:
{selected_solution}

Outcome: {outcome}
Confidence: {majority_vote_ratio}

# What to extract
Prefer reusable solution patterns, theorem cues, invariants, decomposition strategies, sanity checks,
or common failure modes.

# Requirements
- Extract ONE experience item.
- The experience should be generalizable and reusable across similar tasks.
- Focus on a method, diagnostic, trigger pattern, pitfall, or decision rule.
- Avoid copying problem-specific numbers, names, or phrasing unless they express a true general rule.
- Do not summarize the whole solution or restate the full problem.
- Do not write generic advice like "be careful" or "think step by step" without a concrete trigger
  and action.
- Be concise but specific enough to retrieve later.

# Field Semantics
- `intent`: a short retrieval cue describing when this memory should be recalled.
- `title`: a compact human-readable label for the memory.
- `trigger`: a brief condition or pattern indicating when the memory applies.
- `experience`: the reusable content itself, ideally 2-4 short bullets joined into a concise
  paragraph or bullet-style string.
- `tags`: optional short tags such as `algebra`, `elimination`, `pitfall`, `sanity-check`.

# Good Memory Criteria
- It should help on future tasks that are similar in structure, not just this exact problem.
- It should be actionable: a future model should know what to do or what to avoid.
- It should preserve conditions or assumptions when they matter.

# Output Format (strict JSON)
{
  "intent": "short retrieval cue",
  "title": "short memory title",
  "trigger": "when this memory should be used",
  "experience": "2-4 short bullet-style statements or one compact reusable note",
  "tags": ["optional", "tags"]
}

```

Extractor prompt for free-form answer tasks.

```

# Role
You are a Memory Extractor model. You observe a model's solution attempt and extract one reusable
experience item.

# Context
Problem:
{question_with_options}

Model Solution:
{selected_solution}

Outcome: {outcome}
Confidence: {majority_vote_ratio}

# What to extract
For multiple-choice tasks, prefer discriminative cues, elimination rules, contrastive signals between
options, or common traps that cause selecting the wrong option.

# Requirements
- Extract ONE experience item.
- The experience should be generalizable and reusable across similar tasks.
- Focus on a method, diagnostic, trigger pattern, pitfall, or decision rule.
- Avoid copying problem-specific numbers, names, or phrasing unless they express a true general rule.
- Do not summarize the whole solution or restate the full problem.
- Do not write generic advice like "be careful" or "think step by step" without a concrete trigger
  and action.
- Be concise but specific enough to retrieve later.

# Field Semantics
- `intent`: a short retrieval cue describing when this memory should be recalled.
- `title`: a compact human-readable label for the memory.
- `trigger`: a brief condition or pattern indicating when the memory applies.
- `experience`: the reusable content itself, ideally 2-4 short bullets joined into a concise
  paragraph or bullet-style string.
- `tags`: optional short tags such as `algebra`, `elimination`, `pitfall`, `sanity-check`.

# Good Memory Criteria
- It should help on future tasks that are similar in structure, not just this exact problem.
- It should be actionable: a future model should know what to do or what to avoid.
- It should preserve conditions or assumptions when they matter.

# Output Format (strict JSON)
{
  "intent": "short retrieval cue",
  "title": "short memory title",
  "trigger": "when this memory should be used",
  "experience": "2-4 short bullet-style statements or one compact reusable note",
  "tags": ["optional", "tags"]
}

```

Extractor prompt for multiple-choice tasks.

Case Study: Memory Recovers a Global-Constraint Error

Problem. Let $\triangle ABC$ be a right triangle with $\angle A = 90^\circ$ and $BC = 38$. There exist points K, L inside the triangle such that

$$AK = AL = BK = CL = KL = 14.$$

The area of quadrilateral $BKLC$ is $n\sqrt{3}$. Find n .

Evaluation Signal

Setting	Correct	Majority	Parsed answers
No Memory	0/8	98	98, 318, 49, 98, 98, 98, 98, 49
TTML	6/8	104	104, 104, 98, 104, 104, 104, 55, 104

No-memory hard **Recovered majority** **GT: 104**

Retrieved Memory Used by TTML

Shoelace Formula for Area Constrained by Trigonometry. Use a coordinate system with A at the origin, express internal points using geometric constraints, derive a symbolic area formula via the Shoelace formula, and substitute the resulting trigonometric or algebraic constraints.

coordinate geometry **shoelace** **global constraints**

No Memory: Local Construction Trap

Step 1: Fix a local equilateral orientation

The no-memory trajectory places AKL first and implicitly locks the side lengths of the whole triangle:

$$A = (0, 0), \quad K = (14, 0), \quad L = (7, 7\sqrt{3}).$$

Step 2: Derive vertices from the local picture

Using $BK = 14$ and $CL = 14$, it sets

$$B = (28, 0), \quad C = (0, 14\sqrt{3}).$$

Step 3: Fail the global-constraint check

This constructed triangle has

$$BC^2 = 28^2 + (14\sqrt{3})^2 = 1372, \quad BC = 14\sqrt{7} \neq 38.$$

The trajectory notices the inconsistency but continues instead of re-solving under the given hypotenuse $BC = 38$.

Step 4: Return the wrong area

Shoelace on the inconsistent vertices gives

$$\text{Area}(BKLC) = 98\sqrt{3}, \quad \text{<answer>98</answer>}.$$

Failure mode. The model solves a locally valid but globally incompatible geometry. The missing operation is to keep $BC = 38$ active until the final area is determined.

TTML: Memory Preserves the Global Constraint

Step 1: Keep the right-triangle constraint active

TTML keeps the side lengths symbolic instead of fixing a local orientation:

$$A = (0, 0), \quad B = (b, 0), \quad C = (0, c), \quad b^2 + c^2 = 38^2 = 1444.$$

Step 2: Encode equal-distance constraints

From $AK = BK = 14$ and $AL = CL = 14$,

$$K = (b/2, y_K), \quad L = (x_L, c/2).$$

Step 3: Use the equilateral substructure

Since $AK = AL = KL = 14$, triangle AKL is equilateral, giving

$$y_K = c - \frac{\sqrt{3}}{2}b, \quad x_L = b - \frac{\sqrt{3}}{2}c.$$

Step 4: Combine local and global constraints

Applying $KL = 14$ yields $b^2 + c^2 - \sqrt{3}bc = 196$. Together with $b^2 + c^2 = 1444$,

$$\sqrt{3}bc = 1248 \implies bc = 416\sqrt{3}.$$

Step 5: Shoelace area

For $B = (b, 0)$, $K = (b/2, y_K)$, $L = (x_L, c/2)$, and $C = (0, c)$,

$$\begin{aligned} \text{Area}(BKLC) &= \frac{1}{2} \left| by_K + \frac{bc}{4} - y_K x_L + cx_L - bc \right| \\ &= \frac{bc}{4} = \frac{416\sqrt{3}}{4} = 104\sqrt{3}. \end{aligned}$$

Therefore,

$$n = 104 \quad \text{<answer>104</answer>}.$$

Interpretation. The retrieved memory supplies a reusable strategy—coordinate geometry plus Shoelace area under geometric constraints. This strategy helps TTML avoid the no-memory trajectory's local-construction trap, keep $BC = 38$ active throughout the derivation, and recover the correct majority answer.

Figure 8: A no-memory-hard AIME 2025 example. Without memory, all eight no-memory samples are incorrect and the majority answer is 98. With TTML, the retrieved coordinate-geometry/Shoelace memory leads to a constraint-preserving derivation, producing six correct samples and majority answer 104.

Case Study: Memory Converts Low-Consensus Experience into Stable Correctness

Problem. Find the greatest integer n such that

$$x^3 - \frac{n}{6}x^2 + (n - 11)x - 400$$

has roots $\alpha^2, \beta^2, \gamma^2$, where $\alpha, \beta, \gamma \in \mathbb{C}$, and there are exactly seven different possible values of $\alpha + \beta + \gamma$.

Training and Evaluation Signal

Stage	Signal	Evidence
Early epoch 0	$R_{\text{con}} = 0.125$	Low-consensus rollout.
Final eval.	8/8 correct	Majority answer 132 ; all final parsed answers are 132.

Early low consensus Final stable GT: 132

Early Memory Later Used by TTML

Distinct sums of square roots and zero-sum condition. When a cubic has roots $\alpha^2, \beta^2, \gamma^2$ and the number of possible signed sums of α, β, γ is constrained, use sign symmetry to identify a zero-sum choice and then apply Vieta's formulas to the squared roots.

sign symmetry Vieta symmetric sums

Early Epoch: Weak Consensus, Useful Experience

Step 1: The rollouts do not agree

With eight sampled trajectories, the early consensus score is

$$R_{\text{con}} = 0.125 = \frac{1}{8}.$$

The early stage therefore does not provide a reliable majority answer for this query.

Step 2: The extractor preserves the structural cue

Even under weak consensus, TTML extracts a reusable rule: the condition “exactly seven sums” should trigger a sign-symmetry analysis of the square roots, rather than direct coefficient guessing.

Step 3: The memory is not an answer cache

The memory stores an action pattern—choose signs so a zero-sum relation holds, then reduce the problem using symmetric sums. It does not store the final answer 132.

Later Epoch: Memory-Guided Stable Solution

Step 1: Use the sign-symmetry memory

The seven-value condition indicates that, for an appropriate choice of signs,

$$\alpha + \beta + \gamma = 0.$$

Step 2: Apply Vieta to the squared roots

Since the cubic has roots $\alpha^2, \beta^2, \gamma^2$,

$$\alpha^2 + \beta^2 + \gamma^2 = \frac{n}{6}, \quad \alpha^2\beta^2 + \beta^2\gamma^2 + \gamma^2\alpha^2 = n - 11.$$

Step 3: Reduce by symmetric sums

Let $s = \alpha\beta + \beta\gamma + \gamma\alpha$. From $\alpha + \beta + \gamma = 0$,

$$\alpha^2 + \beta^2 + \gamma^2 = -2s, \quad \alpha^2\beta^2 + \beta^2\gamma^2 + \gamma^2\alpha^2 = s^2.$$

Therefore,

$$s = -\frac{n}{12}, \quad s^2 = n - 11.$$

Step 4: Eliminate the intermediate variable

Substituting $s = -n/12$ gives

$$\frac{n^2}{144} = n - 11 \implies n^2 - 144n + 1584 = 0.$$

Hence $n = 12$ or $n = 132$, and the greatest admissible value is

132 <answer>132</answer>.

Interpretation. This case illustrates training-time memory correction. The early epoch supplies only a weak consensus signal, but TTML turns the trajectory into a reusable sign-symmetry/Vieta memory. After memory accumulation, the same structural cue guides the executor to a stable final prediction: all eight final samples parse to the correct answer 132.

Figure 9: A training-process AIME 2026 example. An early epoch has low self-consistency ($R_{\text{con}} = 0.125$), so no reliable early answer set is displayed. TTML nevertheless extracts a reusable sign-symmetry memory, and the final evaluation solves the query with eight correct parsed answers and majority answer 132.