TIMA: Test-Time Internalization for Agentic Memory

Xiaohang Sui[1], **Yongjian Fu**[*1], **Yizhe Zhao**[1], **Sheng Yue**[2], **Ju Ren**[1,3]
[1]Department of Computer Science and Technology, Tsinghua University
[2]School of Cyber Science and Technology, Sun Yat-sen University
[3]State Key Laboratory of Internet Architecture
{suixh26, zhao-yz24}@mails.tsinghua.edu.cn,
{fuyongjian, renju}@tsinghua.edu.cn,
yuesh5@mail.sysu.edu.cn

Abstract

Agentic memory is critical for accumulating and reusing experience in LLM agents across long interaction horizons and related tasks. While latent-memory approaches offer a compact representation of such experience without incurring additional context overhead, existing methods are designed to encode transient in-context states or rely on pre-trained modules that remain fixed at test time to inject experiential representations. Consequently, an effective mechanism for continually refining reusable latent memory from execution experience is still missing in current agents. To this end, we propose TIMA, a framework for test-time internalization that enables agents to acquire and update latent memory during interaction. TIMA couples a memory-augmented agent loop with an online-updated *Internalizer* and an *Internalizer Bank* for retrieval and continual refinement across episodes. Through self-supervised updates derived from execution trajectories, TIMA progressively encodes task-relevant solving patterns into compact latent memory, enabling both within-episode adaptation and cross-episode transfer. Extensive experiments across eight benchmarks show that TIMA consistently outperforms strong baselines, surpassing A-Mem by up to 26.3% and latent-memory baselines such as MemGen and Titans by up to 4.47%–15.96%. Further analysis indicates that the learned latent memory remains stable under continual updates, generalizes well out of distribution, and exhibits interpretable clustering structure across *Internalizers*.

1 Introduction

Large Language Model (LLM)-based agents have recently demonstrated strong capabilities in complex problem solving and real-world interaction [44, 21, 14]. However, many agentic tasks depend not only on reasoning over the current context, but also on the ability to maintain an informative *memory* [12] that retains and refines reusable experience across extended interactions and related tasks. In practice, this remains a major challenge, as agent memory is typically bounded by a finite context window and lacks a persistent mechanism for compact experience accumulation [50, 7]. Consequently, useful information from prior interactions is often not retained in a form that can be reliably transferred to subsequent tasks. These limitations highlight the need for memory mechanisms that support long-horizon reasoning, cross-task adaptation, and behavioral consistency in multi-turn environments.

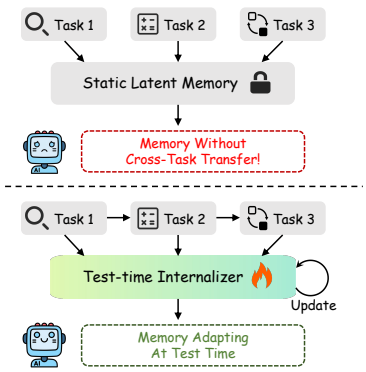


Figure 1: Overview of TIMA.

*Corresponding Author.

Context-based methods [38, 15, 3, 28, 46, 51] store past interactions or condensed experience as natural-language text and reuse them by appending retrieved content to the input prompt. While intuitive and interpretable, such approaches consume the model's limited context budget [5], and their reliance on textual representations makes it difficult to compactly capture abstract task-solving patterns [36]. Latent-memory methods instead encode information as continuous representations in the model's hidden space. However, existing latent memory systems [39, 27, 1, 32, 33] primarily model task-specific states and factual associations, and thus fall short of distilling reusable procedural solving patterns from accumulated agent trajectories.

Several recent works [6, 35, 48] attempt to address this gap by augmenting agents with learnable soft tokens during execution. Although such methods move beyond explicit text-based memory, their memory modules remain fixed at test time, preventing newly generated solving experience from being incorporated into the latent memory itself. This reveals a more fundamental limitation of existing latent-memory methods: while previously encoded experience can be retrieved and reused, ongoing interaction experience cannot be transformed into updated internal representations. Consequently, memory remains a static artifact of pre-training rather than a continually evolving substrate for adaptation during deployment. A complementary line of research on test-time learning (TTL) [53, 24, 10] has shown promise in enabling models to adapt on the fly. However, existing TTL objectives are developed primarily for static language modeling [13, 1] and do not naturally extend to complex agentic workflows, where supervision arises from trajectories that interleave observations, reasoning traces, and actions rather than a homogeneous text stream. Above observations pose a key challenge to us: *How can an LLM agent acquire latent memory that is continuously refined during deployment through a self-supervised signal tailored to agentic workflows?*

To address these challenges, we propose TIMA (**T**est-time **I**nternalization of **M**emory for **A**gents), a framework that equips LLM agents with a lightweight, continuously updated latent memory evolved through test-time learning. At the core of TIMA is the *Internalizer*, a lightweight latent memory module that produces a short sequence of memory tokens at each turn and is refined online through a self-supervised signal derived from agent execution. Its parameters are updated continuously at test time by minimizing the conditional negative log-likelihood of the agent's next composite action, yielding a learning signal grounded in the interleaved structure of the agent's own trajectories rather than in standard language modeling targets. Unlike prior latent-memory modules that remain fixed during deployment, the *Internalizer* continues to evolve with the live task stream and progressively internalizes reusable solving patterns, as shown in Figure 1. Furthermore, to efficiently organize accumulated experiences, TIMA maintains an *Internalizer Bank*, which supports both retrieval of the most suitable *Internalizer* and adaptive updating with new internalized knowledge.

We evaluate TIMA on four task families covering eight benchmarks with multiple backbones, spanning mathematical reasoning, code generation, multi-hop question answering, and long-horizon interactive agents. Across these settings, TIMA consistently outperforms strong baselines, surpassing the textual-memory baseline A-Mem by up to 26.3% on ALFWorld [22] and latent-memory baselines such as MemGen and Titans by up to 4.47%–15.96%. Further analysis indicates that the learned latent memory remains stable under continual updates, generalizes well out of distribution, and exhibits interpretable clustering structure across *Internalizers*. Ablations over key design choices further show that each component contributes consistent gains.

2 Related Work

2.1 Memory Mechanisms for LLM Agents

LLM agent memory has been studied along three complementary directions. Context-based methods store past interactions as natural-language memories and reuse them by appending retrieved content to the prompt [15, 3]. Recent systems further organize memories into multi-level hierarchies [15, 47] and improve retrieval via adaptive routing [41], outcome-driven proxy reasoning [26], or event-centric indexing [11]. A second line internalizes experience into parameters, either by distilling successful trajectories from stronger models into smaller task-adaptive agents [2, 45] or by encoding experience in external parameterized modules [25, 31].

Latent-memory approaches take a different route by encoding information as continuous representations in the model's hidden space. Early works [18, 33] maintain compressive latent memories with self-updating mechanisms. In the agent setting, concurrent efforts [48, 6] extend the idea of

soft reasoning [39] to memory augmentation by injecting learnable latent token sequences during execution, thereby constructing continuous memory from the agent's current state. While these methods move beyond explicit text-based memory, they typically rely on externally pre-trained modules whose parameters remain frozen at deployment, limiting their ability to adapt to the evolving task distributions encountered at test time.

2.2 Test-time Learning

Test-time learning (TTL) adapts models at inference by updating parameters with supervision induced from test inputs. Tent [30] introduced entropy minimization as a fully test-time adaptation objective, while later work developed more conservative variants to mitigate catastrophic forgetting [49]. In language modeling, TTT layers [24] replace attention with a learned hidden state updated by a self-supervised objective for each test sequence, and TLM [10] enables rapid domain adaptation by minimizing input perplexity. Titans [1] further extends this direction with a neural memory module driven by surprise metric. However, these TTL objectives are primarily designed for static sequence modeling or single-turn reasoning. Agent trajectories interleave observations, reasoning traces, and actions, inducing dependencies that differ fundamentally from standard language modeling. This mismatch limits the direct applicability of existing TTL formulations to agent settings. TIMA addresses this gap by equipping agents with latent memory that is continually refined through a test-time learning objective derived from their own execution trajectories.

3 Preliminaries

Agentic Workflow. We consider an LLM agent that engages in multi-turn interaction with an external environment to solve a given task. At each turn t, the agent takes the task instruction q, all previous actions $a_{<t}$, and environmental observations $o_{<t}$, first produces a latent thinking process $z_t \sim \pi_\theta(\cdot \mid q, a_{<t}, o_{<t})$, and then executes an action $a_t \sim \pi_\theta(\cdot \mid q, a_{<t}, o_{<t}, z_t)$, where π_θ denotes the policy parameterized by LLM weights θ. We denote $\hat{a}_t = (z_t, a_t)$ as the composite action at turn t. The environment then returns an observation $o_t \in \mathcal{O}$. This process repeats until a terminal condition is met, yielding an interaction trajectory $\tau = (q, \hat{a}_1, o_1, \hat{a}_2, o_2, \ldots, \hat{a}_T, o_T)$.

Agent Memory System. To augment LLM agents with persistent knowledge that supports long-horizon reasoning, memory-augmented agents [48, 38, 15, 1] introduce an external memory $\mathcal{M}$ that persists across tasks. At each turn, the agent conditions on both past trajectory $\tau_{<t}$ and the memory information retrieved from $\mathcal{M}$ with $\tau_{<t}$:

$$\hat{a}_t \sim \pi_\theta\big(\cdot \mid \tau_{<t}, \texttt{Retrieve}(\mathcal{M}, \tau_{<t})\big) \tag{1}$$

Existing systems instantiate the memory module $\mathcal{M}$ in a variety of forms, including external textual knowledge stores that preserve explicit past interactions [3, 15, 38], and continuous memories that encode experience into parameters [31, 37] or latent states [33], together with a corresponding retrieval function $\texttt{Retrieve}(\cdot)$ tailored to the underlying memory representation. Building on this view, we instantiate agent memory as a retrievable and continually evolving latent module that compactly internalizes task-solving patterns across interactions.

4 Methodology

We propose TIMA, a framework that enables LLM agents to acquire and refine reusable latent memory through test-time learning. As illustrated in Figure 2, TIMA comprises three tightly coupled components. First, it employs a memory-augmented agent loop, in which latent memory is injected at each interaction turn to inform the agent's subsequent reasoning and action (§4.1). Second, it introduces an *Internalizer*, a lightweight module that maps the agent's evolving interaction state to latent memory tokens and is updated online through a self-supervised test-time learning objective defined over the agent's self-generated trajectories (§4.2). Third, it maintains an *Internalizer Bank*, which stores, retrieves, and updates task-relevant *Internalizer*s across episodes, thereby enabling accumulated experience to persist and transfer beyond a single interaction (§4.3).

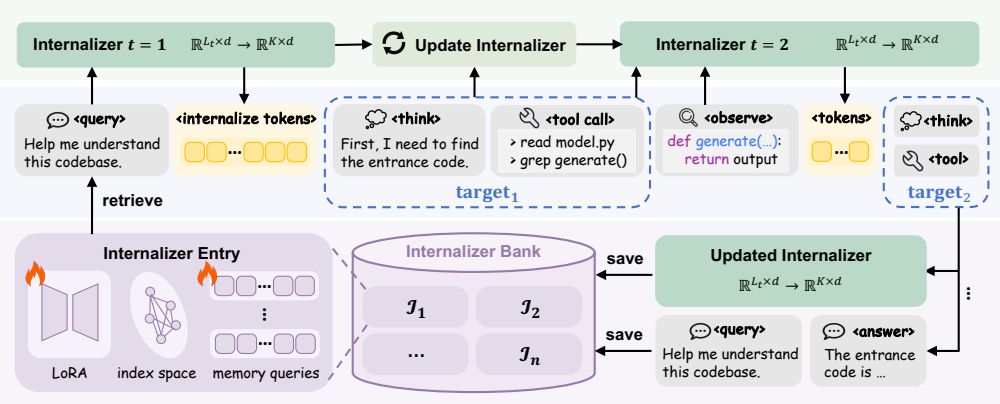


Figure 2: **The TIMA framework.** For each incoming task, TIMA retrieves an *Internalizer* $\mathcal{I}$ from the *Internalizer Bank* (bottom) and uses it to produce latent memory tokens that augment the agent's working loop (middle). After each turn, $\mathcal{I}$ is updated online through a test-time learning objective defined over the agent's own trajectory (top), and the updated parameters are written back to the bank for reuse on subsequent tasks.

4.1 Memory-Augmented Agent Loop

We consider an agent that interacts with the environment in a Think→Act→Observe loop [44]. At each turn t, upon receiving a new observation o_t from the environment (or the initial task prompt q at the start of the episode), TIMA invokes the *Internalizer* to generate a latent memory segment that encodes compact experience relevant to the current interaction state. This memory segment is then incorporated into the current context, and the agent produces its next-step output conditioned on the resulting memory-augmented context.

Concretely, let $\mathbf{h}_{<t} = (\hat{a}_1, o_1, \ldots, \hat{a}_{t-1}, o_{t-1})$ denote the interaction history up to the t-th observation, where each composite action $\hat{a}_t = (z_t, a_t)$ consists of a reasoning trace z_t and an executable action a_t. Given the task query q and a sliding window over the recent interaction history, the *Internalizer* $\mathcal{I}_\phi$ produces K latent memory tokens:

$$m_t = \mathcal{I}_\phi(q, \mathbf{h}_{<t}[-w:]) \in \mathbb{R}^{K \times d}. \tag{2}$$

where d denotes the hidden dimension of the backbone LLM, and $\mathbf{h}_{<t}[-w:] = (\hat{a}_{t-w}, o_{t-w}, \ldots, \hat{a}_{t-1}, o_{t-1})$ denotes a sliding window over the recent interaction history, with w being the window size. These memory tokens are then incorporated into the model input, conditioning the prediction of the next composite action:

$$\hat{a}_t \sim \pi_\theta\big(\cdot \mid q, \mathbf{h}_{<t}, m_t\big) \tag{3}$$

The memory tokens m_t encode internalized solving experience stored in the *Internalizer* retrieved for the current query.

After the agent produces $\hat{a}_t$, TIMA updates the *Internalizer* parameters ϕ using the resulting trajectory as a self-supervised learning signal. Through this test-time adaptation process, the *Internalizer* progressively captures task-relevant solving patterns, allowing the latent memory to evolve alongside the agent's ongoing interaction.

4.2 The *Internalizer*

Architecture. The *Internalizer* is a lightweight external module that encodes the agent's current interaction context into a fixed-length latent memory segment. Given the input token sequence, it produces K continuous memory tokens in the same embedding space as the backbone LLM. The *Internalizer* remains compact relative to the backbone, ensuring that both memory generation and test-time updates can be performed efficiently.

Test-Time Learning Objective. The key mechanism underlying TIMA is that the *Internalizer* is updated online as interaction unfolds. After each turn, the agent produces a composite action $\hat{a}_t =$

(z_t, a_t) conditioned on the memory-augmented context $(q, \mathbf{h}_{<t}[-w:], m_t)$. We use the resulting reasoning-and-action trajectory as a self-supervised signal to update the *Internalizer*: intuitively, a more effective latent memory should render the agent's subsequent reasoning and action more predictable. Formally, we minimize the conditional negative log-likelihood:

$$\mathcal{L}_{\text{TTL}}^{(t)}(\phi) = -\log \pi_\theta\big(\hat{a}_t \mid q, \mathbf{h}_{<t}, m_t(\phi)\big) \tag{4}$$

where $m_t(\phi)$ emphasizes the dependence of the memory tokens on the current *Internalizer* parameters. The backbone parameters θ remain frozen, while gradients are applied only to the *Internalizer* parameters ϕ. This objective drives the *Internalizer* to encode task-relevant solving patterns into its latent memory output, thereby improving the agent's ability to generate appropriate reasoning and actions for related tasks.

Entropy-Aware Objective Selection. Not all interaction turns provide equally reliable learning signals. Turns exhibiting high predictive uncertainty are more likely to induce noisy gradients, which may in turn compromise previously internalized experience. Inspired by entropy-based sample filtering in TTA [19], we therefore adopt a selection mechanism that prioritizes low-entropy turns during updating.

We compute the average token-level entropy of the agent's output over the think-and-act segment $\hat{a}_t$:

$$E_t(\phi) = -\frac{1}{|\hat{a}_t|} \sum_{j=1}^{|\hat{a}_t|} \sum_{v \in \mathcal{V}} \pi_\theta(v \mid c_{<t}, m_t(\phi), \hat{a}_t^{<j}) \log \pi_\theta(v \mid c_{<t}, m_t(\phi), \hat{a}_t^{<j}) \tag{5}$$

where $\mathcal{V}$ is the vocabulary and $c_{<t} = (q, \mathbf{h}_{<t}[-w:])$. We define a turn-level weight:

$$S_t = \exp\big(E_0 - E_t(\phi)\big) \cdot \mathbf{1}\big[E_t(\phi) < E_0\big] \tag{6}$$

where E_0 is a determined entropy threshold. Turns with entropy above E_0 are discarded; those below contribute with weight proportional to their prediction confidence. The filtered objective is:

$$\mathcal{L}_{\text{TTL}}^{\text{filtered}}(\phi) = \sum_{t=1}^{T} S_t \cdot \mathcal{L}_{\text{TTL}}^{(t)}(\phi) \tag{7}$$

Update Procedure. We update the *Internalizer* using a momentum-based rule that balances incorporating new experience against preserving prior knowledge, where the update takes the form:

$$\begin{aligned} u &\leftarrow \mu u + (1-\mu)\, \nabla_\phi \mathcal{L}_{\text{TTL}}^{\text{filtered}}(\phi), \\ \phi &\leftarrow (1 - \eta\omega)\, \phi - \eta\lambda\, u. \end{aligned} \tag{8}$$

where u denotes the momentum-averaged update direction. μ controls momentum strength, with larger μ retaining more past update directions and making the *Internalizer* evolve more conservatively. ω controls forgetting through parameter decay. η and λ jointly control the update magnitude.

4.3 Internalizer Bank

The *Internalizer Bank* $\mathcal{B} = \{(\mathcal{Q}_i, \phi_i)\}_{i=1}^{N}$ maintains a collection of *Internalizer* instances, each associated with a query set $\mathcal{Q}_i$ containing all task instructions it has previously served. The bank enables experience to persist and transfer across episodes.

Retrieval. When a new task query q arrives, TIMA retrieves the most relevant *Internalizer* by comparing the query q with the entry key of each *Internalizer* entry:

$$\mathbf{k}_i = \frac{1}{|\mathcal{Q}_i|} \sum_{q' \in \mathcal{Q}_i} \mathcal{E}(q'), \qquad i^* = \arg\max_i \operatorname{sim}(\mathcal{E}(q), \mathbf{k}_i)\,, \qquad \tilde{\phi} \leftarrow \phi_{i^*} \tag{9}$$

where $\mathcal{E}(\cdot)$ is the embedding encoder and $\operatorname{sim}(\cdot,\cdot)$ denotes cosine similarity. The retrieval mechanism ensures $\tilde{\phi}$ carries internalized experience from prior tasks with a similar structure, providing the agent with a warm-started memory augmentation.

Update and Initialization. After the agent completes the current task and the *Internalizer* has been updated via TTL, the current query is registered into the associated query set: $\mathcal{Q}^* \leftarrow \mathcal{Q}^* \cup \{q\}$. The updated parameters $\tilde{\phi}$ replace the previous version in $\mathcal{B}$, ensuring the bank progressively accumulates solving experience.

When the bank is empty, the first task is handled by a freshly initialized *Internalizer*, so memory emerges solely from the agent's own test-time interaction. As the bank grows, new queries increasingly benefit from accumulated experience. To reduce cold start in practice, we initialize the bank with a small held-out task set, group tasks by query similarity, and use each cluster to briefly prime a dedicated *Internalizer*. This grounds its memory tokens in representative trajectories and provides each entry with a coarse domain-specific prior before deployment on the live task stream.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate TIMA across four task families. For (i) *Mathematical Reasoning*, we use GSM8K [4] and MATH [8]. For (ii) *Code Generation*, we use KodCode [40] and BigCodeBench [52]. For (iii) *Multi-Hop Question Answering*, we use HotpotQA [42] and 2WikiMultihopQA [9], which require retrieval-conditioned reasoning over chains of supporting passages. And for (iv) *Multi-Turn Interactive Tasks*, we use ALFWorld [22], a text-based household simulator that tests long-horizon grounded action planning, and Webshop [43], an online shopping simulator that tests instruction-conditioned navigation and purchase decisions under user specifications.

Baselines. We compare TIMA against baselines spanning four categories. (i) *Vanilla agents* prompt the backbone without any external memory, including direct inference and a CoT-prompted variant [34]. (ii) *Textual memory* inserts retrieved interaction history as natural language in the context, including RAG [16] and A-Mem [38]. (iii) *Latent Memory* methods include MemGen [48] leveraging a tuned memory weaver and the test-time surprise-updated Titans [1]. We additionally compare our method with Search-R1 [14] as (iv) *RL-Augmented Memory* on multi-hop QA tasks.

Implementation Details. We instantiate TIMA on two backbones of different scales, *Qwen2.5-7B-Instruct* [20] and *Llama3.2-3B-Instruct* [29], with identical decoding settings across all methods. The *Internalizer* is instantiated as Qwen2.5-1.5B equipped with LoRA adapters and a set of trainable memory query tokens, followed by a projection layer that aligns its outputs with the hidden space of the backbone model. For the test-time learning objective we fix the number of memory tokens to $K = 8$ and the sliding window size to $w = 1$; the entropy threshold E_0 in Eq. 6 is cold-started for the first 100 update steps on each task stream and is then set to the running p_{50} quantile of the observed turn-level entropies. The *Internalizer Bank* is seeded with 300 successful trajectories per benchmark, taken from AgentBank [23] for interactive tasks and from the official training split for static tasks, and strictly disjoint from the test set. We group them into 20 clusters by query similarity, and each cluster initializes one `seed` entry that provides a coarse domain prior and serves as a stable anchor that is never directly overwritten. When a `seed` entry is retrieved for a new task, we copy it and update the copy through test-time learning on that task, which yields a `spec` entry that accumulates concrete test-time experience; a retrieved `spec` entry is updated in place. Deployment thus begins with a coarse domain prior rather than a cold-start bank, and specialized experience grows without eroding the seeds. Further details on hyperparameters and the full algorithm are deferred to Appendix A.

5.2 Main Results

Table 1 presents the main results on six reasoning benchmarks across two backbones, Qwen2.5-7B-Instruct and Llama3.2-3B-Instruct, while Table 2 reports results on the ALFWorld interactive agent benchmark. We highlight four key observations from these results.

(1) TIMA consistently outperforms all baselines across task families. Table 1 shows that existing baselines exhibit clear trade-offs. Textual-memory approaches, including RAG and A-Mem, remain consistently limited across most benchmarks, indicating that explicit memory injection alone is insufficient for compactly capturing reusable solving patterns. Latent-memory baselines such as Titans and MemGen are markedly stronger overall, underscoring the effectiveness of latent memory

Table 1: **Main results on reasoning benchmarks across two backbones.** Accuracy (%) on math (GSM8K, MATH), code (KodCode, BigCodeBench), and multi-hop QA (2Wiki, HotpotQA). **Bold** denotes the best result per column; a dash (–) indicates methods not evaluated on that benchmark. Search-R1 is a retrieval-specialized RL baseline that is applicable only to multi-hop QA.

Method	**Math**		**Code**		**Multi-Hop QA**		**Avg.**
	GSM8K	MATH	KodCode	BigCodeBench	2Wiki	HotpotQA	
Qwen2.5-7B-Instruct							
Vanilla	90.33	72.60	52.39	36.99	21.61	16.74	48.44
CoT	89.90	73.80	51.61	38.45	23.82	18.11	49.28
RAG	88.43	71.20	48.68	35.09	27.53	30.96	50.31
A-Mem	89.73	76.40	51.02	39.47	39.02	33.69	54.89
Search-R1	-	-	-	-	40.06	41.84	40.95
MemGen	91.60	79.40	65.92	40.64	39.61	38.56	59.29
Titans	89.30	70.60	53.55	36.11	30.64	27.07	51.21
TIMA$_{w/o\ ttl}$	89.60	75.80	59.49	41.52	38.58	36.12	56.85
TIMA	**93.47**	**80.40**	**66.31**	**43.27**	**41.04**	**43.03**	**61.25**
Llama3.2-3B-Instruct							
Vanilla	75.90	45.40	30.28	21.64	21.31	14.61	34.86
CoT	77.00	47.80	29.89	23.25	19.14	15.48	35.43
RAG	73.97	42.80	29.40	19.74	23.83	26.68	36.07
A-Mem	76.93	50.20	32.13	24.17	29.80	28.63	40.31
Search-R1	-	-	-	-	31.52	30.87	31.20
MemGen	81.87	54.80	**40.90**	30.85	30.59	28.92	44.66
Titans	78.93	42.20	31.16	23.68	26.59	24.25	37.80
TIMA$_{w/o\ ttl}$	79.27	49.60	35.73	31.29	29.45	27.85	42.20
TIMA	**83.43**	**56.20**	39.73	**33.04**	**33.65**	**32.04**	**46.35**

Table 2: **Performance on ALFWorld and WebShop (Qwen2.5-7B).** For ALFWorld, we report success rate (%) on the six sub-task types (*pick*, *look*, *clean*, *heat*, *cool*, *pick2*) and the overall average. For WebShop, we report both the average reward score and the average success rate (%). **Bold** denotes the best result per column.

Method	**ALFWorld**							**WebShop**	
	Pick	Look	Clean	Heat	Cool	Pick2	Total	Score	Succ.
Qwen2.5-7B-Instruct									
Vanilla	11.9	5.2	10.3	19.6	4.9	16.1	10.9	26.1	7.6
CoT	50.8	51.6	20.5	30.4	24.4	32.2	37.2	45.4	19.7
MemGen	**93.2**	**69.0**	74.4	69.6	**85.4**	58.1	76.3	70.6	54.9
A-Mem	64.4	60.3	48.7	52.2	41.5	29.0	51.8	53.2	31.4
TIMA	91.5	67.2	**82.1**	**73.9**	**85.4**	**64.5**	**78.1**	**73.4**	**58.7**

representations for reasoning tasks. MemGen is the strongest among them and remains competitive across math, code, and multi-hop QA. However, this advantage is not uniformly sustained across settings. In contrast, TIMA consistently improves upon these baselines across nearly all benchmarks, with gains of up to 4.47% over MemGen and 15.96% over Titans. On Qwen2.5-7B, it achieves the best result on all six benchmarks; on Llama3.2-3B, it ranks first on five of six and remains competitive on KodCode. Relative to the vanilla backbone, TIMA improves the average accuracy by 12.81% and 11.49% on Qwen2.5-7B and Llama3.2-3B, respectively. These results indicate that online memory refinement through test-time internalization yields broader and more robust gains than either textual memory or fixed latent memory alone.

(2) The margin over TIMA w/o TTL isolates the contribution of continual updating. Across both backbones, the full model consistently outperforms TIMA w/o TTL by 1.7%–6.9% on individual benchmarks, with an average improvement of roughly 4.3%. Since the two variants differ only in

whether self-generated trajectories are used to perform test-time updates, this gap directly reflects the contribution of continual internalization. In particular, warm-start retrieval from the *Internalizer Bank* captures only part of the benefit, while online updating provides a further and consistent gain.

(3) Offline-trained latent memory (MemGen) is the strongest non-TTL baseline but still trails. MemGen substantially outperforms textual-memory approaches such as RAG and A-Mem. Nevertheless, TIMA outperforms MemGen on most benchmarks, with the largest gain appearing on HotpotQA under Qwen2.5-7B, where it improves accuracy by 4.47%. More broadly, TIMA shows its clearest advantages on benchmarks such as BigCodeBench, 2Wiki, and HotpotQA, suggesting that online latent-memory refinement is particularly beneficial in settings where task-solving patterns are less stable and must adapt to the test-time distribution.

(4) TIMA transfers cleanly to long-horizon interactive agents. Beyond single-query reasoning, Table 2 evaluates ALFWorld and WebShop, where each episode requires a grounded multi-step action sequence. On ALFWorld, TIMA achieves the best overall performance, improving over A-Mem by 26.3% and over MemGen by 1.8%. On WebShop, TIMA likewise attains the strongest results, exceeding MemGen 2.8 points in score and 3.8% in success rate. The gains over MemGen on ALFWorld are concentrated on sub-tasks that require compositional procedural knowledge: TIMA improves *clean*, *heat*, and *pick2* by 7.7%, 4.3%, and 6.4%, respectively, while remaining near parity on simpler sub-tasks. Across both tables, the consistent gains support a unified view: self-supervised test-time updates from the agent's own trajectories strengthen the mapping from states to reasoning and action in a domain-adaptive manner, with the largest benefits arising where fixed offline memory cannot anticipate test-time solving patterns.

5.3 Analysis

Online Learning Dynamics. Figure 4 (left) tracks cumulative accuracy across the test stream on HotpotQA and GSM8K. TIMA reaches a higher accuracy plateau than all baselines and maintains a stable gap throughout the stream: after the initial warm-up, it stabilizes at roughly 45% on HotpotQA, outperforming TIMA w/o TTL and all other baselines. Since TIMA and TIMA w/o TTL share the same pre-warmed *Internalizer bank* and differ only in whether test-time updates are applied, this persistent gap is directly attributable to continual TTL rather than retrieval alone: online internalization provides gains beyond what a fixed bank can offer. In contrast, Titans remains near or below the vanilla baseline on both benchmarks despite updating its neural memory at test time, which we attribute to the mismatch between its token-level surprise objective and our next-action augmentation objective for agent success. As shown in Figure 4 (right), the *Internalizer Bank* grows sub-linearly and begins to saturate at later steps. As the stream progresses, newly arriving queries increasingly match existing *Internalizer*s rather than spawning new ones, allowing the bank to specialize its entries in place instead of expanding indefinitely, a behavior that emerges naturally from test-time interaction without any explicit capacity cap.

Forgetting and Low-Entropy Selection Analysis. A central concern for any continuously updated memory is whether new experience overwrites prior knowledge. To probe this, we partition the test stream into five blocks and track block-level accuracy as the *Internalizer* is updated over subsequent blocks, with and without the low-entropy selection objective. Figure 5 reports the resulting retention curves on multiple datasets. With low-entropy selection (solid lines), every block only performs minor accuracy drops (e.g., 45.6% $\rightarrow$ 43.7% on HotPotQA) over four additional update rounds; without it (dashed lines), the same blocks shows significant forgetting over the same horizon (e.g., 44.7% $\rightarrow$ 36.9% on HotPotQA), and the decay steepens as more updates are accumulated.

Both variants are evaluated from the same TIMA state before any cross-block update, isolating the gap as a direct effect of the update rule rather than retrieval or initialization. We attribute this separation to entropy acting as a noise filter: high-entropy trajectories provide ambiguous signals whose gradients accumulate destructively across rounds, while filtering them restricts updates to confident, reusable patterns aligned with the knowledge already stored in the *Internalizer*. This retention holds across all three task families, suggesting that the mechanism generalizes beyond a single benchmark.

Memory Analysis. Figure 3 (right) contrasts the geometry of queries with that of their memory-token outputs. Queries of different types occupy well-separated regions in embedding space, and each region is handled by a distinct *Internalizer*, so that after test-time updating, the bank has diversified

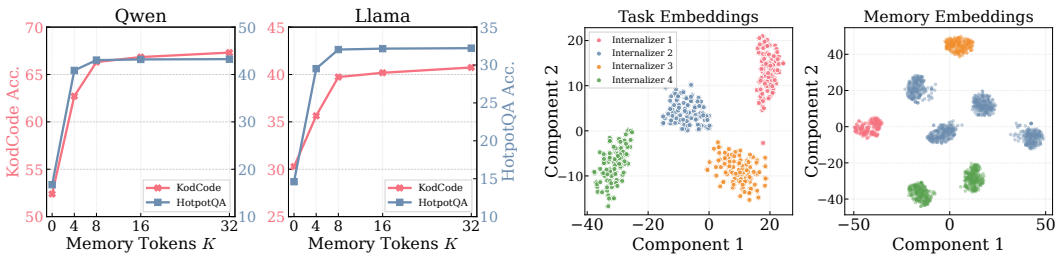


Figure 3: **(Left) Memory token count ablation.** KodCode (red) and HotpotQA (blue) performance under $K \in \{0, 4, 8, 16, 32\}$ on *Qwen2.5-7B-Instruct* and *Llama3.2-3B-Instruct*. **(Right) Memory specialization on HotpotQA.** t-SNE projections of query embeddings and their memory-token outputs within on HotpotQA, colored by the *Internalizer* each query is routed to.

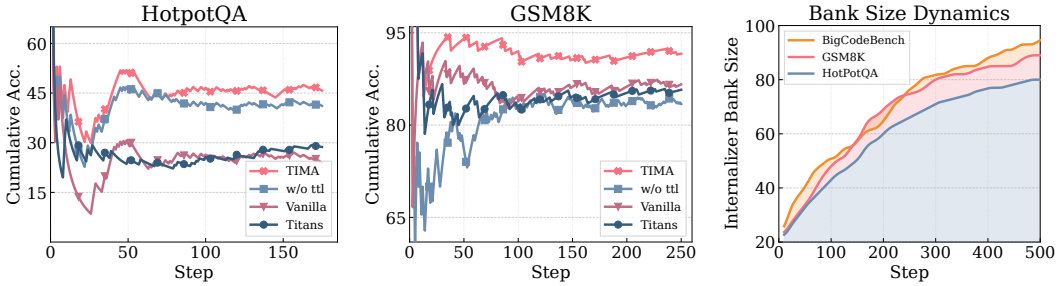


Figure 4: TIMA maintains a persistent accuracy plateau while its *Internalizer Bank* grows sub-linearly. **(Left, Middle)** Cumulative accuracy at each step on HotpotQA and GSM8K for TIMA, TIMA w/o TTL, Direct inference, and Titans. **(Right)** *Internalizer Bank* size at each step on BigCodeBench, GSM8K, and HotpotQA.

across query types with each *Internalizer* specialized to its own slice of the query distribution. The memory outputs on the right preserve this separation, and within some colors they break into additional sub-clusters, suggesting finer distinctions learned inside each *Internalizer*. This embedding-level view matches the specialization-in-place behavior in Figure 4 that bank growth saturates once a small set of Internalizers covers the query-type structure of the stream, and further test-time updates refine each Internalizer internally rather than spawn new entries.

Ablation on Memory Tokens. Figure 3 (left) sweeps the number of memory tokens K produced by the *Internalizer*, from $K = 0$ (no latent memory) up to $K = 32$, on the two representative benchmarks KodCode and HotpotQA. Setting $K = 0$ collapses TIMA to the Vanilla baseline on both backbones, attributing the Table 1 gains to the latent memory tokens. The first few tokens already deliver most of the benefit: moving from $K = 0$ to $K = 4$ raises HotpotQA accuracy by 24.1 points on Qwen, suggesting that even a very compact memory captures the dominant solving-pattern signal. Beyond $K = 8$, accuracy continues to rise monotonically as K grows, indicating that TIMA keeps benefiting from additional memory capacity at diminishing marginal returns. Additional ablation studies are provided in Appendix E.

6 Conclusion

We introduced TIMA, a test-time internalization framework that gives LLM agents a memory that adapts online to the current test stream. Our method maintains a small bank of *Internalizers* that compress agent trajectories into latent memory tokens and are updated at test time with low-entropy selection. Across our benchmark suite and different backbone configurations, TIMA consistently outperforms textual-memory, offline-trained latent-memory, and RL-augmented baselines, while remaining stable under continual updates and generalizing across task families.

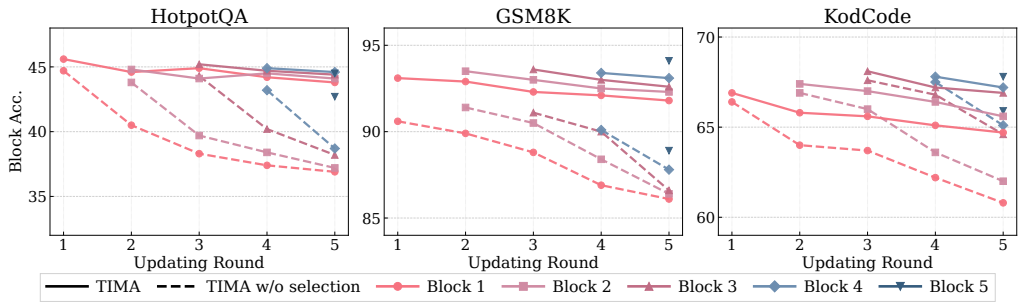


Figure 5: Block-wise accuracy on HotpotQA, GSM8K, and KodCode as the *Internalizer* undergoes rounds of test-time updates on subsequent blocks. Each curve is one of five blocks; the marker on round i is the block's first evaluation (evaluated immediately after TTL on that block), and later rounds evaluate the same block after TTL on subsequent blocks. Dashed lines use the same TTL objective without entropy filtering.

Acknowledgement

This research was supported in part by the National Natural Science Foundation of China under Grant No. 62432004, 6260070575, the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under Grant No. JYB2025XDXM122, the Postdoctoral Innovative Talent Support Program under Grant No. BX2026290, and by a grant from the Guoqiang Institute, Tsinghua University.

References

[1] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

[2] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023. URL https://arxiv.org/abs/2310.05915.

[3] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

[4] 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. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

[5] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. Lightmem: Lightweight and efficient memory-augmented generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=dyJ0GWpjJB.

[6] Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. Latentmem: Customizing latent memory for multi-agent systems, 2026. URL https://arxiv.org/abs/2602.03036.

[7] Zihong He, Weizhe Lin, Hao Zheng, Fan Zhang, Matt W. Jones, Laurence Aitchison, Xuhai Xu, Miao Liu, Per Ola Kristensson, and Junxiao Shen. Human-inspired perspectives: A survey on ai long-term memory, 2025. URL https://arxiv.org/abs/2411.00489.

[8] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

[9] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625,

Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

[10] Jinwu Hu, Zitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=iCYbIaGKSR.

[11] Yuyang Hu, Jiongnan Liu, Jiejun Tan, Yutao Zhu, and Zhicheng Dou. Memory matters more: Event-centric memory as a logic map for agent searching and reasoning, 2026. URL https://arxiv.org/abs/2601.04726.

[12] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026. URL https://arxiv.org/abs/2512.13564.

[13] Jonas Hübotter, Sascha Bongni, Ido Hakimi, and Andreas Krause. Efficiently learning at test-time: Active fine-tuning of LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=NS1G1Uhny3.

[14] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

[15] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25961–25970, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1318. URL https://aclanthology.org/2025.emnlp-main.1318/.

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[18] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024. URL https://arxiv.org/abs/2404.07143.

[19] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International conference on machine learning*, pages 16888–16905. PMLR, 2022.

[20] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[21] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

[22] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

[23] Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories, 2024. URL https://arxiv.org/abs/2410.07706.

[24] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=wXfuOj9C7L.

[25] Jihoon Tack, Jaehyung Kim, Eric Mitchell, Jinwoo Shin, Yee Whye Teh, and Jonathan Richard Schwarz. Online adaptation of language models with a memory of amortized contexts. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=RIfgKCknTu.

[26] Jiejun Tan, Zhicheng Dou, Liancheng Zhang, Yuyang Hu, Yiruo Cheng, and Ji-Rong Wen. Memsifter: Offloading llm memory retrieval via outcome-driven proxy reasoning, 2026. URL https://arxiv.org/abs/2603.03379.

[27] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient KV cache compression through retrieval heads. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tkiZQlL04w.

[28] Yimin Tang, Yurong Xu, Ning Yan, and Masood Mortazavi. Enhancing long context performance in llms through inner loop query mechanism, 2024. URL https://arxiv.org/abs/2410.12859.

[29] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[30] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

[31] Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 53764–53797. Curran Associates, Inc., 2024. doi: 10.52202/079017-1703. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/60960ad78868fce5c165295fbd895060-Paper-Conference.pdf.

[32] Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 74530–74543. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/ebd82705f44793b6f9ade5a669d0f0bf-Paper-Conference.pdf.

[33] Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. Memoryllm: Towards self-updatable large language models, 2024. URL https://arxiv.org/abs/2402.04624.

[34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

[35] Wenyi WU, Zixuan Song, Kun Zhou, Yifei Shao, Zhiting Hu, and Biwei Huang. Towards general continuous memory for vision-language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=YaQnKRtTdh.

[36] Wenyi Wu, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. Auto-scaling continuous memory for gui agent, 2025. URL https://arxiv.org/abs/2510.09038.

[37] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

[38] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=FiM0M8gcct.

[39] Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23336–23351, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1137. URL https://aclanthology.org/2025.acl-long.1137/.

[40] Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6980–7008, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

[41] Shannan Yan, Jingchen Ni, Leqi Zheng, Jiajun Zhang, Peixi Wu, Dacheng Yin, Jing Lyu, Chun Yuan, and Fengyun Rao. Adamem: Adaptive user-centric memory for long-horizon dialogue agents, 2026. URL https://arxiv.org/abs/2603.16496.

[42] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

[43] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/82ad13ec01f9fe44c01cb91814fd7b8c-Paper-Conference.pdf.

[44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[45] Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent lumos: Unified and modular training for open-source language agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.670. URL https://aclanthology.org/2024.acl-long.670/.

[46] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context LLM with multi-conv RL-based memory agent. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=k5nIOvYGCL.

[47] Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems, 2025. URL https://arxiv.org/abs/2512.18746.

[48] Guibin Zhang, Muxin Fu, and Shuicheng YAN. Memgen: Weaving generative latent memory for self-evolving agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vI56m4Iu4e.

[49] Qingyang Zhang, Yatao Bian, Xinke Kong, Peilin Zhao, and Changqing Zhang. COME: Test-time adaption by conservatively minimizing entropy. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=506BjJ1ziZ.

[50] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024. URL https://arxiv.org/abs/2404.13501.

[51] Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=XY8AaxDSLb.

[52] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 66602–66656, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a6a90bcc2aa470c3871b2d39a67d26e8-Paper-Conference.pdf.

[53] 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. Ttrl: Test-time reinforcement learning, 2025. URL https://arxiv.org/abs/2504.16084.

A TIMA Algorithm

Algorithm 1 TIMA: Test-Time Internalization for Agentic Memory

Require: Task query q; frozen LLM π_θ; Internalizer Bank $\mathcal{B} = \{(\mathcal{Q}_i, \phi_i, \texttt{type}_i)\}$ where $\texttt{type}_i \in \{\texttt{seed}, \texttt{spec}\}$; entropy threshold E_0; window w; momentum μ; update scale λ; decay ω; lr η
Ensure: Final answer; updated bank $\mathcal{B}$

// Phase 1: Retrieve Internalizer

Compute $k_i = \frac{1}{|\mathcal{Q}_i|} \sum_{q' \in \mathcal{Q}_i} \mathcal{E}(q')$ for each entry; $\quad i^* = \arg\max_i \, \mathrm{sim}(\mathcal{E}(q), k_i)$

$\tilde{\phi} \leftarrow \phi_{i^*}$

Initialize momentum buffer $u \leftarrow 0$

// Phase 2: Memory-Augmented Agent Loop

Initialize history $\mathbf{h}_0 = \emptyset$

for turn $t = 1, 2, \ldots$ until terminal **do**

  $m_t = \mathcal{I}_{\tilde{\phi}}(q,\ \mathbf{h}_{<t}[-w:]) \in \mathbb{R}^{K \times d}$

  Sample $\hat{a}_t \sim \pi_\theta(\cdot \mid q, \mathbf{h}_{<t}, m_t)$

  Compute entropy $E_t(\tilde{\phi})$ over $\hat{a}_t$

  if $E_t < E_0$ **then**

    $S_t = \exp(E_0 - E_t); \quad \mathcal{L}_t = -S_t \cdot \log \pi_\theta(\hat{a}_t \mid q, \mathbf{h}_{<t}, m_t(\tilde{\phi}))$

    $u \leftarrow \mu u + (1-\mu)\, \nabla_{\tilde{\phi}} \mathcal{L}_t$

    $\tilde{\phi} \leftarrow (1 - \eta\omega)\, \tilde{\phi} - \eta\lambda\, u$

  end if

  Execute a_t; receive o_t; $\quad \mathbf{h}_t \leftarrow (\mathbf{h}_{<t}, \hat{a}_t, o_t)$

end for

// Phase 3: Update Internalizer Bank

if $\texttt{type}_{i^*} = \texttt{seed}$ **then**

  Spawn new specialized entry: $\mathcal{B} \leftarrow \mathcal{B} \cup \big(\{q\},\ \tilde{\phi},\ \texttt{spec}\big)$

else ▷ $\texttt{type}_{i^*} = \texttt{spec}$

  $\mathcal{Q}_{i^*} \leftarrow \mathcal{Q}_{i^*} \cup \{q\}$; $\quad$ recompute centroid k_{i^*}

  $\phi_{i^*} \leftarrow \tilde{\phi}$

end if

return final answer; updated $\mathcal{B}$

B More Experimental Details

For the *Internalizer* updating implementation. During each task, we collect the composite action training objectives at each turn, which are filtered by the entropy threshold and then used to update the *Internalizer*. In practice, we implement the *Internalizer* update with AdamW [17], since its first-moment momentum and decoupled parameter decay align well with Eq. 8 and lead to more stable continual test-time updates. We adopt the corresponding trajectories in AgentBank [23] to seed our *Internalizer Bank* from held-out samples for dynamics tasks while using the official training set for static tasks. Unless otherwise noted, reported numbers are averaged over multiple independent runs. Details of the hyperparameter setting are deferred to Table 3.

The base parameters of Qwen2.5-1.5B are kept frozen. The context embeddings are first projected into the latent space of the *Internalizer*, and K trainable query embeddings are appended to them. After the sequence is processed by the *Internalizer*, the hidden states at the query positions are extracted and mapped by an output projection into K memory tokens in the hidden space of the backbone. The trainable parameters consist of the LoRA adapters, the input and output projections, and the query embeddings.

Table 4: **Main results with standard deviation over five seeds.** Accuracy (%) on math, code, and multi-hop QA benchmarks.

Method	Math		Code		Multi-Hop QA	
	GSM8K	MATH	KodCode	BigCodeBench	2Wiki	HotpotQA
Qwen2.5-7B-Instruct						
A-Mem	89.73±0.67	76.40±0.71	51.02±0.76	39.47±0.95	39.02±1.23	33.69±1.35
Search-R1	–	–	–	–	40.06±1.12	41.84±1.28
MemGen	91.60±0.68	79.40±0.67	65.92±0.55	40.64±0.81	39.61±0.98	38.56±0.90
Titans	89.30±1.25	70.60±1.18	53.55±1.08	36.11±1.36	30.64±1.72	27.07±1.53
TIMA	**93.47**±0.73	**80.40**±0.78	**66.31**±0.62	**43.27**±1.04	**41.04**±1.11	**43.03**±0.88
Llama3.2-3B-Instruct						
A-Mem	76.93±0.72	50.20±0.84	32.13±1.10	24.17±1.09	29.80±1.22	28.63±1.31
Search-R1	–	–	–	–	31.52±1.17	30.87±1.09
MemGen	81.87±0.67	54.80±0.65	**40.90**±0.79	30.85±0.86	30.59±0.99	28.92±1.15
Titans	78.93±1.02	42.20±1.16	31.16±1.41	23.68±1.32	26.59±1.47	24.25±1.78
TIMA	**83.43**±0.82	**56.20**±0.81	39.73±1.03	**33.04**±1.12	**33.65**±1.15	**32.04**±1.20

Table 3: Hyperparameters and implementation settings used in TIMA.

Settings	Hyperparameters
Internalizer	num_latents K = 8 lora_rank = 16 lora_alpha = 32 lora_target_modules = [q_proj, v_proj] lora_dropout = None latent_init_std = 0.02
TTL Update	max_grad_norm = 1.0 optimizer_learning_rate η = 1e-4 optimizer_weight_decay ω = 1e-3 update_strength λ = 1.0 momentum_strength μ = 0.9 entropy_warm_steps = 100
Internalizer Bank	retrieval_encoder = sentence-transformers/all-MiniLM-L6-v2 num_seeds = 20 seed_samples = 300 seed_strategy = kmeans

C Significance Analysis on Main Results

Table 4 reports the means and standard deviations behind the main results over five seeds. TIMA is stable across runs: its standard deviation lies between 0.62 and 1.11 on Qwen2.5-7B-Instruct and between 0.81 and 1.20 on Llama3.2-3B-Instruct, comparable to MemGen (0.55–0.98 and 0.65–1.15) and consistently lower than Titans (1.08–1.72 and 1.02–1.78), the other method that updates its memory at test time. The continual *Internalizer* updates therefore do not add run-to-run instability. Relative to MemGen, TIMA gains between $+0.39$ (KodCode) and $+4.47$ (HotpotQA) points on Qwen2.5-7B-Instruct, and between $+1.40$ and $+3.12$ points on five of the six benchmarks on Llama3.2-3B-Instruct, with the largest margin on HotpotQA for both backbones. The exception is KodCode on Llama3.2-3B-Instruct, where MemGen is ahead by 1.17 points. Relative to Search-R1, TIMA improves 2Wiki and HotpotQA by $+0.98$ and $+1.19$ points on Qwen2.5-7B-Instruct and by $+2.13$ and $+1.17$ points on Llama3.2-3B-Instruct, without any retrieval-specific RL training.

Table 5 gives the same breakdown for the interactive benchmarks. The standard deviations are larger than on static tasks (1.0–2.4 points across all methods), since multi-turn rollouts compound the stochasticity of both the policy and the environment, yet the ordering of methods is unchanged: TIMA

Table 5: **Results on interactive benchmarks with standard deviation.** ALFWorld success rate (%), WebShop score, and WebShop success rate (%) on Qwen2.5-7B-Instruct and **Llama3.2-3B-Instruct**.

Metric	Vanilla	CoT	MemGen	A-Mem	TIMA
Qwen2.5-7B-Instruct					
ALFWorld	10.9±1.1	37.2±1.5	76.3±1.6	51.8±2.0	78.1±1.7
WebShop Score	26.1±1.5	45.4±2.3	70.6±2.0	53.2±2.1	73.4±1.9
WebShop Succ.	7.6±1.0	19.7±1.7	54.9±2.1	31.4±1.9	58.7±1.6
Llama3.2-3B-Instruct					
ALFWorld	8.4±1.2	28.7±1.8	57.9±1.5	38.8±2.3	59.6±1.6
WebShop Score	12.1±1.4	33.9±1.6	58.4±1.9	45.1±2.4	62.7±1.7
WebShop Succ.	4.2±1.2	16.1±1.9	42.3±2.1	25.7±2.0	48.6±2.2

Table 6: **One-sided paired t-tests of TIMA against MemGen and Search-R1 over five seeds.** Positive t-values indicate that TIMA outperforms the baseline; $p < 0.05$ denotes statistical significance.

	vs. MemGen								vs. Search-R1	
Metric	**GSM8K**	**MATH**	**KodCode**	**BigCodeBench**	**2Wiki**	**HotpotQA**	**ALFWorld**	**WebShop**	**2Wiki**	**HotpotQA**
Qwen2.5-7B-Instruct										
t**-value**	3.880	2.341	2.434	7.288	2.675	12.164	2.359	5.990	2.486	2.555
p**-value**	0.009	0.040	0.036	0.001	0.028	<0.001	0.039	0.002	0.034	0.031
Llama3.2-3B-Instruct										
t**-value**	3.268	3.264	−2.612	2.982	6.667	4.349	2.236	5.423	2.536	3.173
p**-value**	0.015	0.015	0.971	0.021	0.001	0.006	0.044	0.003	0.032	0.017

is the best method on every metric for both backbones. On Qwen2.5-7B-Instruct, it improves over MemGen by $+1.8$ on ALFWorld, $+2.8$ on WebShop score, and $+3.8$ on WebShop success rate; on Llama3.2-3B-Instruct the corresponding gains are $+1.7$, $+4.3$, and $+6.3$. The largest gains appear on WebShop success rate, and the 6.3-point margin on Llama3.2-3B-Instruct is about three times the standard deviation of either method. The gap to A-Mem is considerably larger, reaching $+26.3$ on ALFWorld and $+27.3$ on WebShop success rate with Qwen2.5-7B-Instruct.

Table 6 complements these statistics with one-sided paired t-tests over the five seeds ($\mathrm{df} = 4$). Against MemGen and Search-R1, TIMA is significantly better ($p < 0.05$) in all ten comparisons on Qwen2.5-7B-Instruct and in nine of ten on Llama3.2-3B-Instruct, with the largest p-value among the significant results being 0.044. The evidence is strongest on HotpotQA ($t = 12.164$, $p < 0.001$) and BigCodeBench ($t = 7.288$, $p = 0.001$) with Qwen2.5-7B-Instruct, and on 2Wiki ($t = 6.667$, $p = 0.001$) and WebShop ($t = 5.423$, $p = 0.003$) with Llama3.2-3B-Instruct. The comparisons against Search-R1 on 2Wiki and HotpotQA are significant on both backbones. The only comparison in which TIMA is not significantly better is KodCode on Llama3.2-3B-Instruct, where MemGen has a 1.17-point higher mean ($t = -2.612$).

D Experience Transferring Analysis

We acquire the *Internalizer Bank* through our framework on one source dataset and evaluate directly on held-out targets without further adaptation, using Qwen2.5-7B-Instruct for all methods. Two source–target groups are considered: KodCode→{BigCodeBench, HumanEval} for code generation, and HotpotQA→{2Wiki, GPQA} for multi-hop reasoning.

As shown in Figure 6, TIMA leads on both in-domain sources and preserves its edge under moderate shift. Relative to MemGen, the strongest latent-memory baseline, TIMA improves by $+2.3$ points when transferring from KodCode to BigCodeBench and by $+4.9$ points when transferring from HotpotQA to 2Wiki, indicating that the internalized memory retains a recoverable prior across adjacent task distributions rather than overfitting to a single dataset. A few additional observations are also worth highlighting. First, TIMA stays above Vanilla on every one of the six target settings, so keeping the bank active never incurs a measurable transfer cost even when the target drifts far from the source distribution. Second, on the QA side, TIMA remains ahead of Search-R1 across all three targets without any retrieval-specific RL training, suggesting that test-time internalization can match the cross-distribution robustness of a retrieval-specialized policy at a substantially lower training cost.

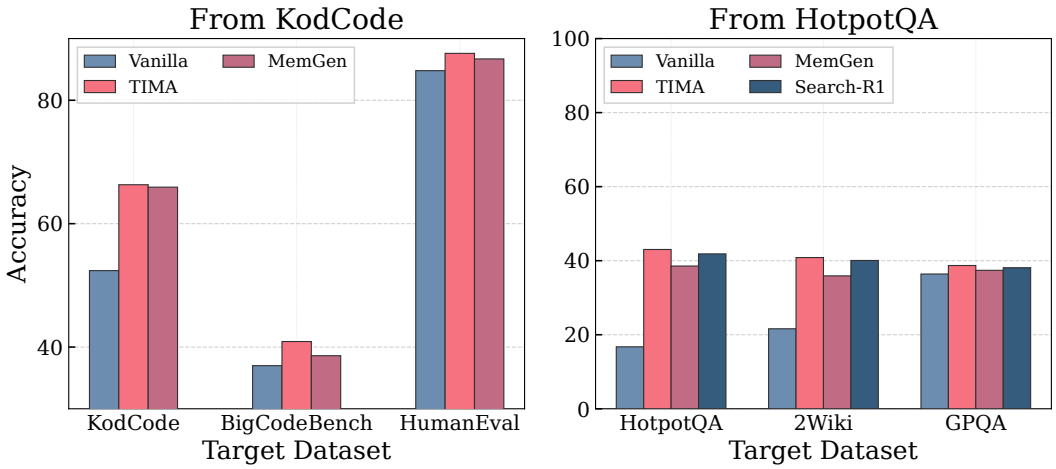


Figure 6: **Experience Transferring Performance.** For TIMA, the *Internalizer Bank* is accumulated on KodCode (left) or HotpotQA (right) and evaluated on held-out targets. TIMA retains its lead across various sets, which underscores the generalization ability of our method.

Table 7: **Effect of LoRA rank r in the *Internalizer*.** Performance on GSM8K, KodCode, and 2Wiki with the *Qwen2.5-7B-Instruct* backbone. Bold marks the rank used in the main experiments.

LoRA Rank	GSM8K	KodCode	2Wiki
$r = 4$	91.63	62.84	38.02
$r = 6$	92.07	63.58	38.74
$r = 8$	92.57	64.92	39.55
$r = 16$ **(Default)**	93.47	66.31	41.04
$r = 24$	<u>93.60</u>	66.52	<u>41.26</u>
$r = 32$	93.57	<u>66.73</u>	41.19

E Ablation Study

E.1 Ablation on LoRA Rank

Table 7 ablates the LoRA rank r used inside the *Internalizer* on three representative datasets (GSM8K, KodCode, 2Wiki), keeping all other hyperparameters fixed at their main-experiment values. Accuracy rises steadily from $r = 4$ to $r = 16$ on every dataset, indicating that the *Internalizer* needs a minimum amount of low-rank capacity to absorb the reusable solving patterns carried by each trajectory. The sensitivity to rank is not uniform across tasks: KodCode and 2Wiki benefit noticeably more from a larger rank than GSM8K does, suggesting that code generation and multi-hop question answering depend more heavily on latent adaptation capacity. Beyond $r = 16$, the curves flatten, and the higher ranks $r = 24$ and $r = 32$ yield only minor, task-specific fluctuations; although the per-column best (underlined) is attained at $r = 24$ or $r = 32$, the gap to $r = 16$ is minor on every dataset. We therefore adopt $r = 16$ as the default setting in the main experiments, trading a negligible accuracy margin for a substantially lower test-time update cost.

E.2 Ablation on Low-Entropy Threshold

Table 8 sweeps the entropy threshold E_0 used by the low-entropy selection rule in Eq 6, expressed as the percentile of the running turn-level entropy distribution at which the threshold is placed. $E_0 = p_{50}$, which keeps the bottom half of the turns for TTL, is the best setting on all three benchmarks, showing that filtering out roughly half of the highest-entropy turns strikes the right balance between the updating signal and noise turns. The two directions away from this peak are not symmetric: raising the threshold to p_{75} hurts more than lowering it to p_{25}, indicating that the destructive effect of admitting additional noisy high-entropy turns outweighs the cost of keeping fewer updates. Disabling

Table 8: **Effect of the entropy threshold E_0 used by low-entropy selection.** E_0 is set to a percentile of the running turn-level entropy distribution.

E_0 Setting	GSM8K	HotpotQA	KodCode
p_{10}	91.87	39.24	63.29
p_{25}	92.83	41.38	65.04
p_{50} **(Default)**	**93.47**	**43.03**	**66.31**
p_{75}	92.77	40.70	64.65
No filtering	91.57	37.10	62.51

Table 9: **Backbone-only online LoRA TTL ablation.** To isolate the effect of online updating from the additional 1.5B auxiliary parameters, *TTL on Backbone* applies online LoRA test-time learning directly to the backbone with the same objective, while removing the memory-token injection mechanism. Best results are in bold.

Method	GSM8K	KodCode	HotpotQA	ALFWorld
Vanilla	90.33	52.39	16.74	10.9
TTL on Backbone	89.67	55.11	24.83	50.6
TIMA	**93.47**	**66.31**	**43.03**	**78.1**

filtering altogether produces the weakest result on every benchmark, and the resulting HotpotQA accuracy aligns with the w/o selection setting in Figure 5.

E.3 Ablation on Auxiliary Parameters

Table 9 separates the effect of online updating from that of the additional 1.5B auxiliary parameters. Applying the same low-entropy TTL objective directly to the backbone with online LoRA, without memory-token injection, improves KodCode by 2.72 points, HotpotQA from 16.74 to 24.83, and ALFWorld from 10.9 to 50.6, while GSM8K slightly decreases from 90.33 to 89.67. Test-time learning is therefore useful on its own for multi-turn and knowledge-intensive tasks, but it does not account for the full gain: TIMA outperforms the backbone-only variant by $+3.80$ on GSM8K, $+11.20$ on KodCode, $+18.20$ on HotpotQA, and $+27.5$ on ALFWorld. The drop on GSM8K is consistent with online updates altering the pretrained weights that carry the model's reasoning ability, whereas TIMA keeps the backbone frozen and confines adaptation to the *Internalizer*, which supplies memory through latent tokens. The improvement thus comes from combining memory-token injection with test-time internalization rather than from extra parameters or online updating alone.

E.4 Ablation on Internalizer Size

Table 10 replaces the default Qwen2.5-1.5B-Instruct *Internalizer* with the smaller Qwen2.5-0.5B-Instruct on Llama3.2-3B-Instruct. Reducing the *Internalizer* to one third of its default size lowers accuracy by 2.13, 1.07, and 3.80 points on GSM8K, KodCode, and HotpotQA, respectively, indicating that the capacity of the *Internalizer* translates into the quality of the generated latent memory. The drop is largest on HotpotQA, suggesting that multi-hop reasoning relies more on the expressiveness of the memory generator than arithmetic or code generation does. The 0.5B variant introduces fewer additional parameters and lower computational overhead, and it still remains ahead of Titans on all three benchmarks (78.93, 31.16, and 24.25), so it is a practical option when latency is the main concern. We use the 1.5B *Internalizer* by default for the best accuracy.

E.5 Ablation on Context-window Size

Table 11 varies the number of most recent action–observation pairs w provided to the *Internalizer*. Accuracy decreases monotonically as the window grows on all three benchmarks: moving from $w = 1$ to $w = 4$ lowers GSM8K, KodCode, and 2Wiki by 1.00, 1.17, and 0.93 points, respectively, with $w = 2$ falling in between. The *Internalizer* thus draws the most useful signal from the latest interaction step, while earlier pairs lengthen the input and dilute the current step with older context.

Table 10: **Internalizer-size ablation on Llama3.2-3B-Instruct.** We replace the default Qwen2.5-1.5B-Instruct Internalizer with the smaller Qwen2.5-0.5B-Instruct, which introduces fewer additional parameters and lower computational overhead. Best results are in bold.

Internalizer	GSM8K	KodCode	HotpotQA
Qwen2.5-1.5B **(Default)**	**83.43**	**39.73**	**32.04**
Qwen2.5-0.5B	81.30	38.66	28.24

Table 11: **Effect of the context-window size w.** w denotes the number of most recent action–observation pairs used as context for the Internalizer. Best results are in bold.

Context window size (w)	GSM8K	KodCode	2Wiki
1 **(Default)**	**93.47**	**66.31**	**41.04**
2	93.10	65.53	40.70
4	92.47	65.14	40.11

The changes are small in magnitude, so TIMA is not sensitive to this hyperparameter. Since $w = 1$ is both the most accurate and the cheapest setting, we adopt it as the default.

F More Experimental Results

F.1 Forgetting Analysis w/o TTL

To quantify how much of TIMA's stability against forgetting is attributable to continual TTL updates, we further run a counterfactual in which the *Internalizer* is initialized by one TTL pass over Block 1 and then frozen for the remainder of the stream. Figure 7 plots this frozen variant (dashed) alongside the full TIMA (solid). By construction, the two curves coincide at Round 1, which confirms that any subsequent divergence is attributable to the update rule rather than to initialization. Once the *Internalizer* is frozen, accuracy drops sharply as the stream moves to Block 2, and then stabilizes at this reduced level across the remaining blocks. The flatness of the dashed curve after the initial drop is itself informative: it indicates that the loss is not random noise but a systematic mismatch between the frozen memory and the distribution of later blocks, which a frozen *Internalizer* cannot close. In contrast, TIMA's performance is identical to Figure 5, only decline mildly over the same horizon, as continual updates re-align the *Internalizer* with the evolving stream. The gap between the two regimes directly measures the contribution of TTL updates beyond retrieval and initialization.

F.2 More t-sne Visualizations

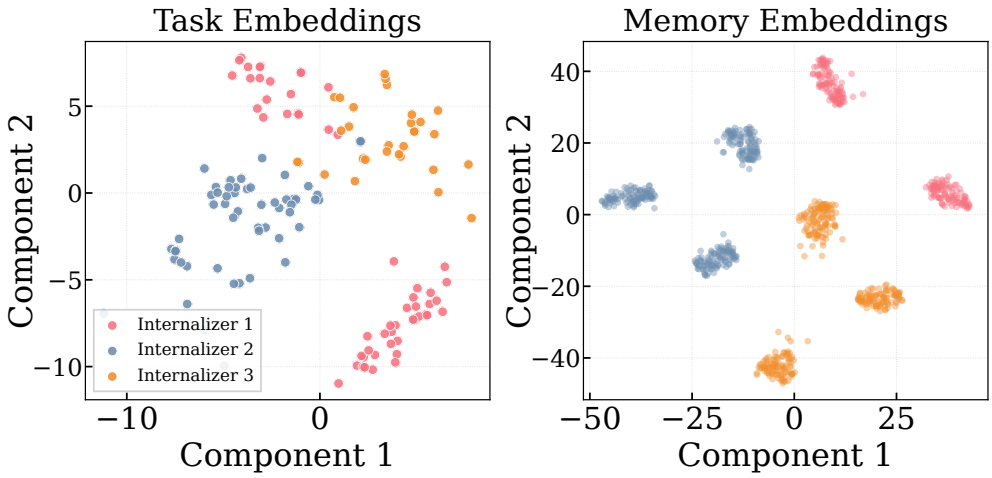


Figure 8: t-SNE visualization on KodCode.

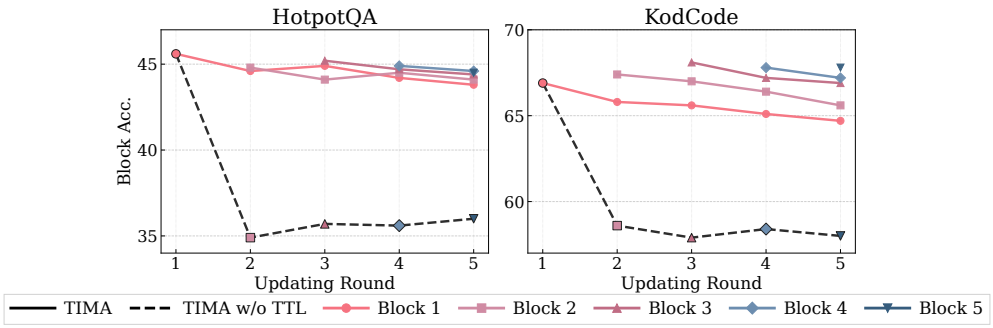


Figure 7: Ablation of TTL mechanism under continual updates and frozen *Internalizer* on HotpotQA and KodCode. Each solid curve tracks one block's accuracy across subsequent TTL update rounds; the dashed curve reports the accuracy of an *Internalizer* frozen after one TTL pass on Block 1 and then evaluated on each successive sample block.

Table 12: **Efficiency comparison on Qwen2.5-7B-Instruct.** For TIMA, Time and Total TFLOPs are reported as generation + update. Param Effic. denotes the accuracy gain over Vanilla per million trainable parameters.

Method	Trainable Params	KodCode					GSM8K					HotpotQA				
		Time (s)	Acc.	Param Effic.	Output Length	Total TFLOPs	Time (s)	Acc.	Param Effic.	Output Length	Total TFLOPs	Time (s)	Acc.	Param Effic.	Output Length	Total TFLOPs
Qwen2.5-7B-Instruct																
Vanilla	0	14.82	52.39	–	325.4	8.74	9.48	90.33	–	213.2	4.97	17.36	16.74	–	387.6	7.35
MemGen	30.82M	7.18	65.92	0.44	141.7	12.45	7.64	91.60	0.04	135.4	9.54	11.29	38.56	0.71	191.0	15.93
Titans	2.17M	10.24	53.55	0.53	206.2	5.64	10.72	89.30	−0.47	219.3	5.02	16.42	27.07	4.76	346.9	6.38
TIMA	13.21M	8.46 (6.69+1.77)	**66.31**	1.05	146.8	14.43 (7.37+7.06)	8.76 (6.55+2.21)	**93.47**	0.24	162.2	9.59 (5.79+3.80)	13.31 (10.58+2.73)	**43.03**	1.99	220.1	10.85 (5.76+5.09)

F.3 Efficiency Analysis

Table 12 reports the average inference time and accuracy on KodCode, GSM8K, and HotpotQA. Overall, TIMA achieves the best accuracy on all three datasets while maintaining moderate runtime overhead. Compared with MemGen, TIMA is slightly slower due to the additional *Internalizer* test-time update, but the overhead remains limited, which is about 15–18% across the three datasets. In return, TIMA improves accuracy by $+1.87$ and $+4.47$ percentage points against the strongest baseline on GSM8K and HotpotQA, respectively. Compared with Vanilla, TIMA is consistently faster, reducing runtime by 42.9% on KodCode and 23.3% on HotpotQA. This suggests that the injected latent memory can guide the backbone toward more focused generation paths, especially for code generation and multi-hop reasoning where unconstrained decoding often leads to longer and less productive trajectories. Titans, despite also performing test-time memory updates, is slower than TIMA on all three datasets and yields substantially lower accuracy, indicating that TIMA provides a better efficiency–performance trade off.

F.4 Self-NLL Objective Analysis

Table 13 examines whether the negative entropy that drives the self-NLL objective is informative about trajectory quality. Across all eight datasets, the negative entropy of a trajectory separates successful from failed trajectories with an AUROC between 0.6633 and 0.7640, well above the chance level of 0.5, and Kendall's τ_b is positive on every dataset (0.2787–0.3411), so lower entropy is consistently associated with a higher chance of success. The association is strongest on WebShop (AUROC 0.7640, $\tau_b = 0.3411$), followed by BigCodeBench and HotpotQA, and weakest on KodCode (0.6633) and GSM8K (0.6794); the two metrics largely agree on this ordering. Entropy is thus an informative but imperfect indicator of correctness, which is why TIMA uses it to select the turns used for updates (Eq. 6) instead of treating every generated turn as supervision.

Table 14 compares updating the *Internalizer* on different subsets of turns. Updating on the accepted turns, our default, is the best on all eight benchmarks, and updating on the rejected turns is the worst on all eight, with the gap between the two ranging from 6.22 points on 2Wiki to 29.8 points on ALFWorld. Since $E_0 = p_{50}$ splits the turns into two roughly equal halves, this gap is not caused by

Table 13: **Correlation between negative entropy and trajectory success.** We report AUROC and Kendall's τ_b on each dataset.

Metric	WebShop	BigCodeBench	GSM8K	MATH	HotpotQA	KodCode	TriviaQA	2WikiMultihopQA
AUROC	0.7640	0.7245	0.6794	0.6834	0.7167	0.6633	0.7017	0.6946
Kendall's τ_b	0.3411	0.3070	0.2787	0.2940	0.3235	0.2892	0.3024	0.2976

Table 14: **Effect of update-turn selection.** Performance (%) of TIMA when the Internalizer is updated on different subsets of turns. *Accepted turns* uses only the turns that pass the low-entropy filter (default, $E_0 = p_{50}$); *Rejected turns* uses only the turns filtered out; *Updates without filtering* uses all turns. Best results are in bold.

Update turns	GSM8K	MATH	KodCode	BigCodeBench	2Wiki	HotpotQA	ALFWorld	WebShop
Accepted turns	**93.47**	**80.40**	**66.31**	**43.27**	**41.04**	**43.03**	**78.1**	**58.7**
Rejected turns	83.90	57.20	48.76	33.67	34.82	28.59	48.3	32.2
Updates without filtering	91.57	73.40	62.51	37.88	38.08	37.10	65.7	49.7

a difference in the number of updates but by which turns are used. Updating on all turns without filtering lies in between and trails the accepted-only setting by 1.90–12.4 points, with the largest deficits on ALFWorld (12.4) and WebShop (9.0), whose long-horizon trajectories contain more turns for noisy updates to accumulate. Updating on rejected turns alone even drops GSM8K and KodCode below the Vanilla results (90.33 and 52.39), whereas updating on accepted turns raises them. These results agree with the AUROC analysis above: low-entropy turns carry a reliable learning signal, while high-entropy turns mostly contribute noise.

G Discussion

G.1 Limitations

Although TIMA shows consistent gains across diverse benchmarks, several limitations remain. First, the current framework is evaluated on a finite set of task families and backbone models, and its behavior under substantially broader agent settings, more heterogeneous tool environments, or rapidly shifting task streams remains to be studied. Second, while the *Internalizer* is lightweight, TIMA still introduces additional test-time computation due to online memory generation and continual updates, which may become non-negligible in latency-sensitive deployment scenarios.

Third, the low-entropy selection in TIMA acts as a noise filter rather than a correctness filter. Entropy reflects how confident the model is in its own output, not whether that output is correct: the AUROC between negative entropy and trajectory success ranges from 0.66 to 0.76 (Table 13), so the filter tends to retain higher-quality turns but cannot exclude confidently wrong ones, which pass the threshold and are reinforced by the self-NLL update. Relatedly, Figure 5 shows that accuracy still declines as test-time updates accumulate; entropy filtering slows this forgetting compared with updating on all turns, but it does not eliminate it. Incorporating signals beyond the model's own confidence, such as environment feedback or verifiable outcomes, and explicitly constraining the drift of the *Internalizer* over long streams are natural directions for future work. Exploring more efficient update mechanisms, more scalable memory organization, and more robust adaptation is an important direction for future work.

G.2 Broader Impacts

This work studies continually updated latent memory for LLM agents. Such a capability may improve the adaptability, efficiency, and long-horizon consistency of agents in complex environments, with potential benefits for interactive assistants and other agentic systems. At the same time, stronger adaptive memory may also introduce risks, since agents may internalize biased or undesirable patterns from ongoing interaction and may be misused in longer-horizon harmful workflows. We therefore believe that deployment of such systems should be accompanied by appropriate safeguards, including monitoring of memory updates, constraints on update sources, and safety evaluation in realistic settings.

H Case Study

Cases from HotPotQA

Case 1

System Prompt

Answer the given multi-hop question. You must reason step by step inside <think> and </think> before every action, and you must think again after each new observation. If you still need evidence, call the search engine with <search> query </search>. The search engine will return documents between <information> and </information>. You may search multiple times. When you have enough evidence, provide only the final short answer inside <answer> and </answer>. For example, <answer> Beijing </answer>.

User Prompt

Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

Assistant Output

[TIMA Tokens]<think> To answer this question, I need to know what program the Apple Remote was originally designed for and if there are other devices that can control it.</think> <search> original program for Apple Remote </search>

Information

Doc 1(Title: Siri Remote) The Siri Remote is a remote control device released in tandem with the 4th Generation Apple TV at an Apple Keynote Address by CEO Tim Cook on October 26, 2015 by Apple Inc. The Remote is designed specifically to work with the always-on Siri capable Apple TV...[truncted]

Assistant Output

[TIMA Tokens]<think> From the information provided, the Apple Remote was originally designed to interact with the Front Row media program on the iSight iMac G5 and is compatible with some later desktop and portable Macintosh computers. </think> <search> devices compatible with Front Row media program </search>

Information

Doc 1(Title: Front Row (software)) Front Row is a discontinued media center software application for Apple's Macintosh computers and Apple TV for navigating and viewing video, photos, podcasts, and music from a computer, optical disc, or the Internet through a 10-foot user interface...[truncted]

Assistant Output

[TIMA Tokens]<think> The information confirms that the Apple Remote was designed to interact with the Front Row media program. The Front Row media program can be controlled by an Apple Remote or the keyboard function keys. </think> <answer> keyboard function keys </answer>

Case 2

System Prompt

Answer the given multi-hop question. You must reason step by step inside <think> and </think> before every action, and you must think again after each new observation. If you still need evidence, call the search engine with <search> query </search>. The search engine will return documents between <information> and </information>. You may search multiple times. When you have enough evidence, provide only the final short answer inside <answer> and </answer>. For example, <answer> Beijing </answer>.

User Prompt

What material did a hairdresser from Yorkshire, England invent that was named by his granddaughter?

Assistant Output

[TIMA Tokens]<think> To answer this question, I need to identify the hairdresser from Yorkshire, England, and the material he invented that was named by his granddaughter. I will start by searching for a notable hairdresser from Yorkshire, England. </think><search> notable hairdressers from Yorkshire, England </search>

Information

Doc 1(Title: Samuel Ironside) Samuel Ironside (9 September 1814-24 April 1897) was an English Methodist missionary in New Zealand, where he became a supporter and signatory of the Treaty of Waitangi...[truncted]

Assistant Output

[TIMA Tokens]<think> The search results do not mention a hairdresser from Yorkshire, England who invented a material named by his granddaughter. I need to narrow down the search to hairdressers specifically and look for any inventions or materials they might have created. </think> <search> hairdressers from Yorkshire, England inventions </search>

Information

Doc 1(Title: List of English inventions and discoveries) English inventions and discoveries are objects, processes or techniques invented...[truncted]

Assistant Output

[TIMA Tokens]<think> The search results still do not provide the specific information I need. I need to look for a hairdresser from Yorkshire, England who invented a material and had a granddaughter involved in naming it. I will try a more specific search. </think> <search> Yorkshire hairdresser invented material named by granddaughter </search>

Information

Doc 1(Title: Starlite) Starlite is a material claimed to be able to withstand and insulate from extreme heat. It was invented by amateur chemist and hairdresser Maurice Ward (1933-2011) during the 1970s and 1980s...[truncted]

Assistant Output

[TIMA Tokens]<think> The search results provide the information I need. Maurice Ward, an amateur chemist and hairdresser from Yorkshire, England, invented Starlite, a material claimed to withstand and insulate from extreme heat. The name Starlite was coined by his granddaughter, Kimberly. </think> <answer> Starlite </answer>

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly describe TIMA's scope as a test-time internalization framework for agentic memory. The claims are supported by the main experiments and analyses on our memory framework.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussion the limitations of our work and propose future directions in Appendix G.1.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.

- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: We do not include theoretical results in this paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We disclose all implementation details in Section 5.1 and Appendix B. We will release our data and code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.

- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We release our code and data in supplemental material and will open-source them upon paper acceptance.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We disclose all implementation details in Section 5.1 and Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We run our experiments multiple times and report the significance analysis of our main results in Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We discuss the efficiency of our method in Appendix F.3 and we run all our experiments on NVIDIA A40 GPUs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research is with the NeurIPS Code of Ethics

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader impacts of our work in Appendix G.2.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our research has no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, assets are properly respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our research does not involve crowdsourcing and research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our research does not involve crowdsourcing and research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: Our research focuses on enhancing LLM agent's memory abilities. The core method development in this research does not involve LLMs as any important components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.