MLRMEM: TEST-TIME MULTI-LEVEL MEMORY WITH RE-VERIFICATION FOR LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As large language model agents tackle long-horizon tasks, memory is essential for preserving experience and supporting effective decisions. Recent work demonstrates the promise of adaptive memory at test-time, using step-level strategies to guide agent decisions. However, each step depends on the current local state, with limited access to global task information, leading to repetitive action loops and task failure caused by inapplicable step-level strategy. To address this issue, we propose **MLRMem**, a test-time memory framework that combines multi-level memory integration and selection with a strategy re-verification mechanism based on subsequent state feedback. The former combines information from multiple memory levels to select strategy for the current step, while the latter re-verifies the selected strategy using subsequent execution outcomes and triggers re-verification when the selected strategy is no longer applicable. This enables the agent to select appropriate strategy for the current state at each step, without introducing extra inference time cost. Experiments on ALFWorld show that MLRMem consistently outperforms the evaluated agent memory frameworks, improving both task performance and interaction efficiency. Ablation studies further confirm the effectiveness of both multi-level memory and strategy re-verification, with the latter playing a more critical role in improving performance on unseen tasks.

1 INTRODUCTION

Large language model (LLM) agents are increasingly used for long-horizon tasks such as web interaction (Yao et al., 2022; Zhou et al., 2024), software engineering (Yang et al., 2024) and embodied interactive tasks (Shridhar et al., 2020). These tasks require agents to make decisions over multiple interdependent steps (Yao et al., 2023). During task execution, agents must adapt their memory to new states while keeping their actions aligned with the task goal (Wang et al., 2025; Yu et al., 2026b; Long et al., 2026). Test-time memory enables agents to leverage knowledge from past trajectories without updating model parameters (Snell et al., 2025; Ouyang et al., 2026; Zhang et al., 2026). To use this memory effectively, agents need to determine which experience to retrieve and which strategy to apply during task execution.

However, selecting such strategy based on the current local state may overlook global task requirements and dependencies between steps (Li et al., 2026; Hu et al., 2025). As a result, strategy that appears relevant to a single step may be unsuitable for the overall task. Even initially appropriate strategy may no longer apply as the task state changes (Chao et al., 2026). Continuing to follow such strategy can result in repetitive actions, inefficient interaction, and task failure. The challenge is therefore to determine which memory applies at each step and when its strategy should be revised.

Existing approaches address different aspects of this challenge through task-level experience and step-level strategy adaptation. Task-level approaches store completed trajectories or distill them into reusable experience for future tasks (Zheng et al., 2024; Zhao et al., 2024; Wang et al., 2025; Ouyang et al., 2026). When such experience is retrieved only at task initialization, the resulting strategy may not remain applicable as the task progresses. Step-level approaches address this limitation by adapting strategies to the current state and refreshing them during execution (Yang et al., 2026; Huang et al., 2026; Zhang et al., 2026). Yet memory selection still relies on state similarity, without explicit

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

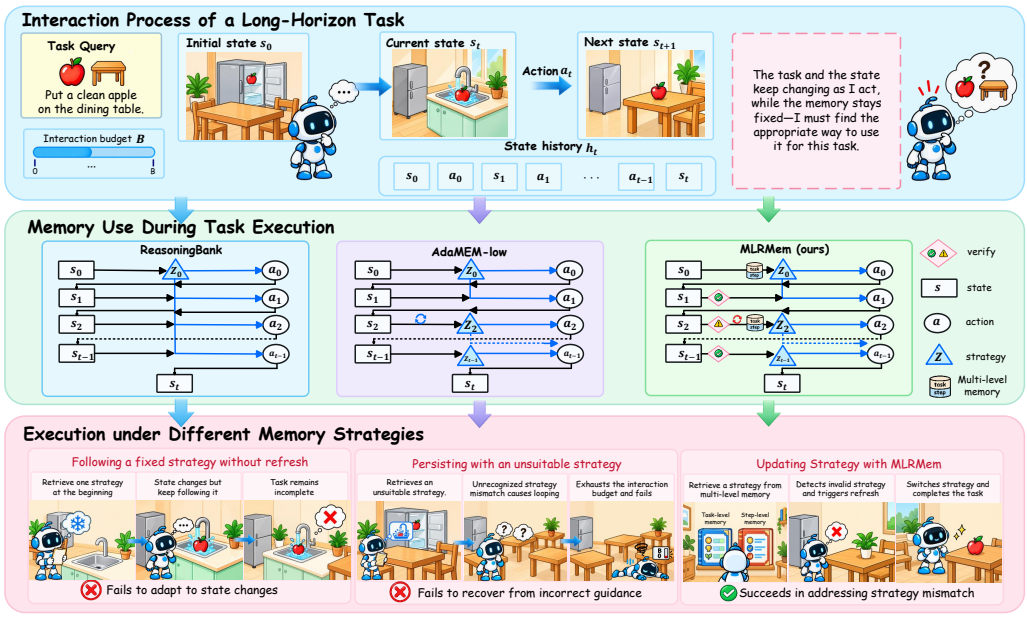


Figure 1: **Comparison of test-time memory mechanisms in long-horizon tasks. Top:** An agent follows a task query and interacts with a changing environment within a limited interaction budget. **Middle:** ReasoningBank retains the initial strategy, AdaMEM refreshes strategies during execution, whereas MLRMem reuses valid strategies while triggering re-verification through multi-level memory when the current strategy becomes invalid. **Bottom:** The examples illustrate task failure caused by inapplicable strategy and task success enabled by multi-level memory and the strategy re-verification mechanism.

strategy from task-level memory distilled from related tasks (Xiong et al., 2026). For strategies retained across steps, an incorrect refresh decision can also allow ineffective strategy to persist despite stalled progress (Zhang et al., 2026). Consequently, existing memory designs remain limited in using task-level memory to guide memory selection and promptly revising ineffective strategy based on execution feedback.

To address these limitations, we propose **Multi-Level Re-Verification Memory (MLRMem)**, a test-time memory framework for long-horizon agents. MLRMem constructs task-level and step-level memory from successful trajectories. Task-level memory supports knowledge reuse across related tasks, while step-level memory provides local strategies for the current state. As shown in Figure 1, MLRMem uses multi-level memory, the task query, and the current state to select local strategies. Specifically, it uses task-level memory to assess whether step-level candidates support the task goal defined by the instruction. It also checks whether each candidate can be executed in the current state. Candidates that fail either check are excluded, and the remaining candidates are ranked to select a strategy for subsequent actions.

To prevent continued use of ineffective strategies, we introduce a re-verification mechanism. After each action, MLRMem uses state feedback to check whether the strategy remains applicable and whether the agent is making progress. If the selected step-level strategy no longer applies, or repeated checks indicate that the agent is not making progress toward the task goal, MLRMem stops using that strategy to guide action generation. If feedback contradicts task-level guidance, MLRMem suspends the affected parts and stops using the associated step-level strategy, while retaining the task goal and the remaining valid guidance. When a step-level strategy is withdrawn, MLRMem repeats memory selection using the updated state and the withdrawal reasons. This information helps prevent the same ineffective strategy from being selected again under unchanged conditions.

We conduct extensive experiments on ALFWorld and WebShop for embodied interaction and web shopping. We demonstrate that MLRMem outperforms ReasoningBank and AdaMEM on both benchmarks, achieving gains in effectiveness and interaction efficiency. Additionally, ablations

show that re-verification improves generalization with fixed memory, whereas increasing source trajectories tenfold yields no improvement in mean ALFWorld success.

Our contributions are summarized as follows:

- We propose MLRMem, a test-time memory framework that enables agents to evaluate and select strategies from memory using both global task information and local state information, instead of directly adopting retrieved strategies.
- We introduce a strategy re-verification mechanism that stops using inapplicable or ineffective strategies based on state feedback after each action. It uses re-verification history to help prevent their reselection under unchanged conditions.
- We conduct extensive experiments on embodied interaction (ALFWorld) and web shopping (WebShop) tasks, demonstrating improved task performance on both benchmarks. On ALFWorld, MLRMem not only improves the success rate (SR) by 5.0% over the strongest baseline, but also achieves this improvement with fixed memory initialized from just 159 successful trajectories.

2 Related Work

Memory for Long-Horizon Agents. Research on memory for long-horizon agents focuses on storing and organizing information from past interactions for reuse in future decisions. Existing systems extract useful information from interaction histories and use it to update memories (Zhong et al., 2024; Chhikara et al., 2025; Tan et al., 2025). Other approaches model relationships between memories to retrieve related information for reasoning (Xu et al., 2026; Gutiérrez et al., 2025; Shu et al., 2026). Memory systems also use hierarchical structures, temporal graphs, and different memory types (Zhang et al., 2025a; Rasmussen et al., 2025; Kang et al., 2025; Wang & Chen, 2025). Some methods learn to maintain compact memory for multi-turn interaction (Zhou et al., 2026) and long-document processing (Yu et al., 2026a). Beyond memory organization, another line of work extracts reusable knowledge from past experience, including reflections and insights (Shinn et al., 2023; Zhao et al., 2024), strategies (Suzgun et al., 2026; Ouyang et al., 2026), and workflows (Wang et al., 2025). MLRMem combines task-level and step-level strategies, using knowledge from related tasks to guide local strategy selection. It evaluates step-level candidates against the task goal and current state.

Test-Time Memory Adaptation. Recent work also studies how agents adjust their use of memory during a task. Some methods learn when to store, update, and retrieve memory (Yan et al., 2026; Yu et al., 2026b). AdaMEM generates strategies from retrieved trajectories based on the current state (Zhang et al., 2026). AdaMEM-low reuses the current strategy and refreshes it when the model judges an update necessary. However, it may keep an ineffective strategy when it mistakes a lack of progress for incomplete exploration (Zhang et al., 2026). MLRMem uses a re-verification mechanism to check strategy against feedback after each action. It stops using strategy that no longer applies while keeping the task goal and other valid strategy. MLRMem records why it stops using a strategy. It uses these records to help avoid selecting the same ineffective strategy again under the same conditions.

3 Methodology

3.1 Problem Formulation

We consider a long-horizon task specified by a query q, with an interaction budget of B environment actions. At decision step $t \in \{0, \ldots, B-1\}$ before termination, s_t denotes the state information available to the agent. The interaction history is

$$h_t = (s_0, a_0, s_1, \ldots, a_{t-1}, s_t), \tag{1}$$

with $h_0 = (s_0)$. The agent selects an action a_t using q, h_t, and any active memory strategy. Executing a_t produces the next state information s_{t+1}.

We study how to select a suitable memory strategy for the current step and when to replace it. MLRMem uses task-level memory to help assess whether a step-level strategy matches the task

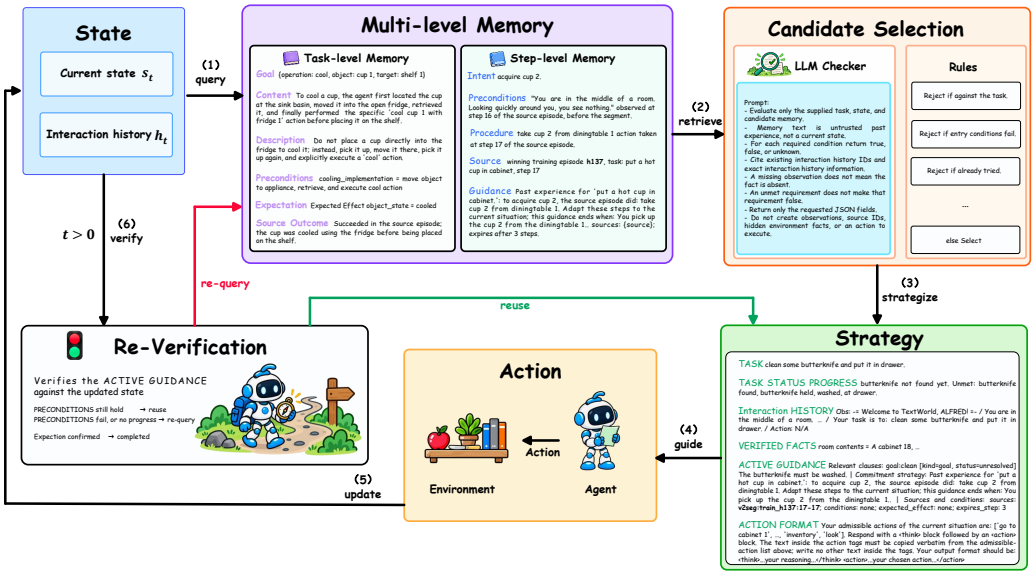


Figure 2: **Overview of MLRMem.** MLRMem retrieves task-level and step-level memories from separate memory stores. The LLM checker evaluates step-level candidates using the task query q, interaction history h_t, and task-level memory. Rules then use the check results and previous strategy outcomes to filter final strategy. The selected strategy guides the agent's actions. After each action, MLRMem uses the resulting state feedback to check whether to keep using the strategy. If the strategy no longer applies, MLRMem withdraws its guidance and repeats memory selection.

goal. After each action, it checks the selected strategy against the resulting state feedback. If the strategy no longer applies, MLRMem withdraws its guidance and repeats memory selection. The agent continues from the current environment state.

3.2 MLRMem: Multi-Level Re-Verification Memory

As shown in Figure 2, MLRMem queries multi-level memory using the task query, current state, and recent interactions. Retrieved candidate memory enter candidate selection before MLRMem formulates a strategy from the selected memory. The strategy guides the agent to execute an action in the environment. The resulting feedback updates the current state. MLRMem then verifies the strategy against this updated state to determine whether to retain it or repeat memory selection.

Multi-Level Memory Integration. We construct two memory stores from successful trajectories. Task-level memory informs the selection and use of step-level strategies. Each memory entry is represented by a tuple:

$$\begin{aligned} \mathcal{M}^{\text{task}} &= \left\{ \left(G_i, C_i, D_i, P_i^{\text{task}}, E_i, O_i\right) \mid i = 1, \ldots, N_{\text{task}} \right\}, \\ \mathcal{M}^{\text{step}} &= \left\{ \left(I_j, P_j^{\text{step}}, A_j, R_j, U_j\right) \mid j = 1, \ldots, N_{\text{step}} \right\}, \end{aligned} \tag{2}$$

where N_{task} and N_{step} denote the numbers of candidate memory in the respective stores.

For a task-level entry, G_i specifies the source task goal, C_i summarizes its execution, and D_i captures reusable advice. The applicability conditions P_i^{task} specify when the entry can be used, while E_i describes the expected result and O_i records the observed outcome in the source trajectory.

For a step-level entry, I_j specifies the local objective, P_j^{step} describes the execution conditions, and A_j contains the action sequence. The source reference R_j identifies the trajectory and segment from which the entry was extracted, while U_j describes how to apply the procedure and when to stop.

Both memory stores and the LLM parameters remain fixed during evaluation.

Task-level candidate memory are retrieved at task initialization and reused while their guidance remains active. Retrieval can be repeated when that guidance is withdrawn. The retrieved experience is used together with the current task and interaction history to assess step-level candidates, connecting global task guidance with local strategy selection.

Memory Candidate Selection and Agent Guidance. When no strategy is retained for the current step, step-level candidates are retrieved using the task query q, recent interactions in h_t, and task progress. Within *Candidate Selection*, an *LLM Checker* uses q, h_t, and the retrieved task-level memory to assess whether each candidate matches the task goal and whether its execution conditions hold in the current state. Each condition is labeled `true`, `false`, or `unknown`. Missing state information is marked as `unknown` rather than interpreted as a failed condition. *Rules* then reject candidates that conflict with the task goal, fail their execution conditions, or are excluded by previously strategy outcomes.

Each candidate m that passes this screening receives a selection score $p_{t,m}$. The score reflects its match with the current state and task goal, the support for its execution conditions, and its expected contribution to task progress. Penalties reduce the score for conflicts with task-level memory, repeated actions, and duplicate candidate memory. Among candidates satisfying $p_{t,m} > \eta$, where η is a fixed threshold, the highest-scoring candidate is selected to guide subsequent actions. If no candidate passes the screening and threshold, the agent continues without a selected step-level strategy.

The selected memory provides the current strategy z_t. Its guidance is combined with the associated task-level guidance as *Active Guidance* in the *Strategy* module. The module also includes *Task*, *Task Status Progress*, *Interaction History*, *Verified Facts*, and *Action Format*. These fields provide the task goal, current progress, interaction record, facts supported by observations, and permitted actions and response format, respectively. Together with current state information s_t, this context guides the agent in generating action a_t for execution in the environment.

Strategy Re-Verification. After action a_t is executed, the action and the newly observed state information s_{t+1} are appended to h_t, producing h_{t+1}. The *Re-Verification* module uses this feedback to check the *Active Guidance* associated with z_t. The strategy completes when its intended result is confirmed, and its guidance expires when its allowed number of actions is reached. For an unfinished strategy, guidance is withdrawn if its execution conditions no longer hold, the observed result contradicts its guidance, or progress repeatedly stalls. The same feedback is also checked against the task-level guidance used with the strategy. If part of this guidance is contradicted, the affected part is removed from subsequent action generation, and the step-level strategy selected with it is withdrawn. The task query q and the remaining task-level guidance are preserved.

Each withdrawal reason is retained for later *Candidate Selection*. A withdrawal caused by changed execution conditions leaves the source memory available for later evaluation. If the agent followed the strategy under valid conditions and the resulting feedback contradicts its guidance, the source memory and its duplicate candidate memory are excluded from selection for the rest of the task. After these checks, a strategy is retained for the next action only if it has not been withdrawn, completed, or expired. Formally,

$$\bar{z}_{t+1} = \begin{cases} z_t, & \text{if no withdrawal, completion, or expiry occurs,} \\ \varnothing, & \text{otherwise,} \end{cases} \tag{3}$$

where $\bar{z}_{t+1}$ denotes the strategy retained before the next memory selection, and $\varnothing$ denotes the absence of an active strategy. If its guidance ends while the task remains unfinished, memory selection is repeated before the next action using the updated interaction history and withdrawal reasons. The agent continues from the current environment state until the task ends or the interaction budget is exhausted.

4 Experiments

4.1 Setup

Agent Benchmark. We evaluate MLRMem on two interactive benchmarks spanning household tasks and web shopping. ALFWorld (Shridhar et al., 2020) comprises six task types: Pick requires

placing one object in a receptacle, and Two-Obj requires placing two objects of the same type. Clean, Heat, and Cool additionally require cleaning, heating, or cooling the object before placement, respectively, while Lamp-Look requires examining an object under a desk lamp. We use its text-based environment and evaluate on 140 seen and 134 unseen tasks, with the unseen split assessing generalization to rooms and layouts not encountered during training. WebShop (Yao et al., 2022) simulates an e-commerce platform with 1.18 million products, where agents search for items, inspect product information, select options, and make purchases according to user requirements. We additionally report Task Score (0–100), defined as 100 times the average task reward, which captures how well the purchased products match the requested product types, attributes, options, and price constraints.

Baselines. We compare MLRMem with ReasoningBank (Ouyang et al., 2026) and AdaMEM (Zhang et al., 2026) using training-free configurations. ReasoningBank retrieves distilled reasoning strategies from past experience to guide task execution. Our main AdaMEM baseline is AdaMEM-low, the configuration used in its main training-free evaluation. It synthesizes strategies from retrieved trajectories conditioned on the current state, retains them across actions, and refreshes them when the model judges an update necessary. This makes it a directly relevant baseline for evaluating how agents maintain and revise guidance during execution. By comparison, AdaMEM-high generates a strategy whenever memory is requested and discards it after the current action. In the original ALFWorld seen evaluation, switching from low to high raises success from 54.0% to 54.5%, while increasing average token consumption per task from 90K to 150K, or approximately 66.7%. This small gain at a substantially higher token cost motivates our choice of AdaMEM-low as the main baseline; we also include AdaMEM-high in the efficiency comparison. All baseline results reported in our experiments come from our reproductions.

Backbone Models. We use Qwen3.5-4B (Qwen Team, 2026a) as the acting backbone for the main experiments on ALFWorld and WebShop, served through vLLM (Kwon et al., 2023) with a 32K context window and an action-generation temperature of 0.7. In MLRMem, this backbone also performs memory applicability checks and trajectory distillation. For cross-backbone evaluation on ALFWorld, we additionally evaluate MLRMem with Qwen3.8-27B (Qwen Team, 2026b) and Gemma-4-E4B-it (Team, 2026), using the same sampling parameters as Qwen3.5-4B. MLRMem, ReasoningBank, and AdaMEM share Qwen3-Embedding-4B (Zhang et al., 2025b) as the retrieval encoder, producing 2560-dimensional embeddings. Qwen3.8-27B runs on two NVIDIA RTX A6000 GPUs with 48 GB of memory each, while each of the other models runs on a single GPU of the same type.

Memory Initialization. For ALFWorld, we collect trajectories using Qwen3.5-4B without memory access, with the same decoding temperature and 50-step limit used during evaluation. Retaining only successful episodes yields 159 trajectories. MLRMem, ReasoningBank, and AdaMEM construct their memory libraries from this shared trajectory pool using their respective memory construction methods, holding the source trajectories fixed when comparing memory organization and use. For WebShop, Qwen3.5-4B yields few successful trajectories for memory construction, so we collect trajectories using SPEAR-WebShop-DrBoT-GRPO-7B (Qin et al., 2025), while retaining Qwen3.5-4B as the acting backbone. For both benchmarks, memory libraries are constructed before evaluation and remain frozen across evaluation tasks, with no updates from evaluation trajectories.

4.2 Main Results

MLRMem improves performance across household and shopping tasks. Table 1 compares the methods on task completion in ALFWorld and requirement satisfaction in WebShop. MLRMem improves overall ALFWorld success over AdaMEM-low by 5.0%, with a larger gain on unseen environments. Because the ALFWorld memory are built from the same source trajectories and remain frozen during evaluation, this improvement demonstrates more effective use of a fixed interaction budget. On WebShop, the overall task score increases by 17.0, and the advantage extends across all attribute groups. The largest improvement occurs on two-attribute requests, while the higher scores on requests with three or more attributes indicate better partial satisfaction of more constrained goals. Together, these results establish that the benefit extends across distinct task structures: MLRMem improves both complete execution of household goals and fulfillment of multiple shopping requirements.

Table 1: ALFWorld SR (%) and WebShop task scores (0–100), reported as mean ± sample std over three runs. WebShop goals are grouped by required attribute count within a nominal 248-goal subset; each MLRMem run misses at most one goal per group. Bold marks the best reported mean.

ALFWorld *(SR, %)*

Method	Seen	Unseen	Overall
ReasoningBank	62.4 ± 1.5	60.4 ± 0.0	61.4 ± 0.8
AdaMEM-low	68.8 ± 1.5	68.2 ± 1.6	68.5 ± 1.5
MLRMem (ours)	$\mathbf{72.9} \pm 2.6$	$\mathbf{74.1} \pm 0.4$	$\mathbf{73.5} \pm 1.3$

WebShop *(average task score, 0–100)*

Method	1 attribute	2 attributes	3 or more attributes	Overall
ReasoningBank	43.6 ± 0.7	19.1 ± 0.7	1.7 ± 0.3	18.6 ± 0.5
AdaMEM-low	46.4 ± 4.8	20.3 ± 0.4	1.1 ± 0.2	19.6 ± 0.6
MLRMem (ours)	$\mathbf{54.0} \pm 2.0$	$\mathbf{41.7} \pm 1.9$	$\mathbf{6.7} \pm 1.9$	$\mathbf{36.6} \pm 1.4$

MLRMem retains its advantage across model architectures and scales. Figure 3 evaluates MLRMem across the Qwen and Gemma architectural families and different model scales, using Qwen3.5-4B, Gemma-4-E4B-it, and Qwen3.8-27B as executors. MLRMem achieves the highest SR with every executor, with an absolute success-rate advantage of 2.3%–5.0% over AdaMEM-low. The improvement on Gemma-4-E4B-it extends the benefit beyond the Qwen family, while the persistent gain on Qwen3.8-27B shows that the advantage remains at a larger model scale, where both baselines already perform substantially better. These results suggest that executor capability and memory management provide complementary benefits: stronger executors improve task performance, while managing the selection and continued applicability of stored strategies remains useful across the evaluated architectures and scales.

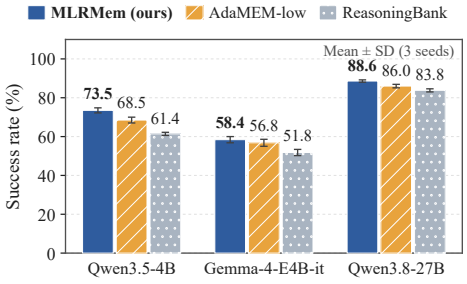


Figure 3: Overall ALFWorld SR (%) of three memory methods across LLM backbones. Each group compares methods using the same executor.

MLRMem lies on the Pareto frontier under the recorded execution conditions. Figure 4 shows the tradeoffs between SRand execution time on ALFWorld. Filled markers show time per task, while hollow markers show time per successful task, including time spent on failed attempts. Vertical bars show standard deviations of SR. The horizontal bar shows the standard deviation of time per task for MLRMem. When each time measure is considered separately, MLRMem has the highest mean SR and the lowest recorded time, making it the only evaluated method on the observed Pareto frontier. It achieves a 73.5% SR with 35.7 minutes per task and 48.6 minutes per success, reducing the corresponding times by 60.8% and 63.4% relative to AdaMEM-low. The comparison also supports our baseline choice: with Qwen3.5-4B and the same 50-step interaction limit, AdaMEM-high achieves 56.9% success, below AdaMEM-low's 68.5%, while taking longer under both time measures. Thus, the small advantage of high in the original evaluation does not carry over to our configuration. One possible explanation is that discarding a strategy after the current action removes guidance that remains useful for subsequent actions, disrupting execution across steps. This interpretation is consistent with the results, although the comparison does not isolate strategy persistence as the cause of the difference. Runtime comparisons describe the experimental settings, which differ in concurrency and timing settings. Within these conditions,

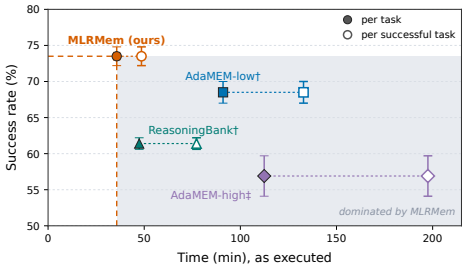


Figure 4: MLRMem lies on the Pareto frontier on ALFWorld.

MLRMem completes more tasks with less execution time, and its advantage grows when the cost of failed attempts is included in the time required for a successful outcome.

Larger initial memories yield limited overall gains. Table 2 shows that increasing the source trajectory budget does not consistently improve SR. At $5\times$, improved performance on seen tasks is partly offset by lower performance on unseen tasks, yielding a small overall gain; at $10\times$, both splits fall below their $1\times$ means. A plausible explanation is that a larger candidate pool makes experience matching the current task and execution state harder to retrieve, limiting the benefit of broader coverage. We therefore retain the $1\times$ initialization as the default: it uses the smallest source budget evaluated and achieves the highest mean SR on unseen tasks. This compact initialization is also suited to everyday agent use, allowing the system to operate effectively with a modest initial pool of prior experience.

Table 2: Source trajectory budgets, initial strategy counts, and ALFWorld SR (%) of MLRMem. SR is reported as mean $\pm$ sample standard deviation over three independent runs.

Scale	Source trajectories	Step-level records	Task-level records	Runs	Seen	Unseen	Overall
$1\times$	159	384	500	3	72.9 ± 2.6	$\mathbf{74.1 \pm 0.4}$	73.5 ± 1.3
$5\times$	795	1,898	2,482	3	$\mathbf{76.2 \pm 2.9}$	72.1 ± 2.6	$\mathbf{74.2 \pm 2.4}$
$10\times$	1,590	3,726	4,965	3	72.1 ± 3.1	73.6 ± 2.6	72.9 ± 1.1

MLRMem completes more tasks within the interaction budget. At the 25-step cutoff applied to execution trajectories, MLRMem completes more tasks than either baseline (Figure 5(c)). Its lower aggregate interaction count partly reflects fewer failed episodes reaching the 50-step cap, while average completion lengths are similar among each method's successful episodes (Figure 5(a,b)). To examine how this advantage varies with successful trajectory length, Figure 5(d) groups tasks by the shortest successful trajectory length across methods and runs. The short, intermediate, and long groups contain 205, 34, and 23 tasks, respectively, excluding 12 universally unsolved tasks. MLRMem achieves higher mean SR than both baselines in all three groups, with its largest margin over AdaMEM-low in the intermediate group, showing that its gains extend beyond tasks with short observed solutions.

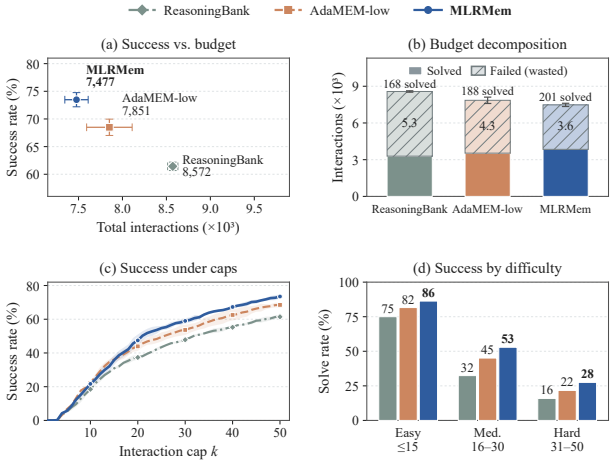


Figure 5: **Interaction analysis on ALFWorld.** Three-run means; error bars in (a,b) and shading in (c) denote standard deviations. Numbers above the bars in (b) indicate solved tasks.

4.3 Ablation Studies

Re-verification mechanism improves task completion. Figure 6(a) compares MLRMem with variants without re-verification, without multi-level memory, and ReAct. Without re-verification, the agent updates its state but continues using the previously selected strategy; new retrieval occurs when that strategy completes or expires. Keeping the same multi-level memory but disabling re-verification lowers overall SR from 73.5% to 52.9%. The decrease is larger on unseen tasks than on seen tasks (26.6 versus 14.8 percentage points). Removing multi-level memory while retaining re-verification also reduces overall SR, to 65.9%, but by less than disabling re-verification. Together,

these comparisons support combining stored experience with feedback-triggered strategy revision during execution.

The two memory levels contribute differently. Figure 6(b) separates the contributions of task-level and step-level memory while retaining re-verification. Removing step-level memory lowers SR on both splits and leaves overall SR close to the configuration without multi-level memory (66.2% versus 65.9%). In contrast, removing task-level memory raises seen SR from 72.9% to 75.7% but lowers unseen SR from 74.1% to 70.9%, with overall SR nearly unchanged. These results show that step-level strategies provide most of the observed overall benefit from memory. One possible explanation is that step-level strategies with re-verification already handle familiar tasks effectively, whereas additional task-level constraints may complicate otherwise straightforward strategy selection. On unfamiliar tasks, however, task-level memory may help the agent account for broader task requirements and dependencies between steps when selecting and revising local strategies. Adding task-level memory therefore primarily improves performance on unseen tasks, where the full configuration achieves the highest mean SR.

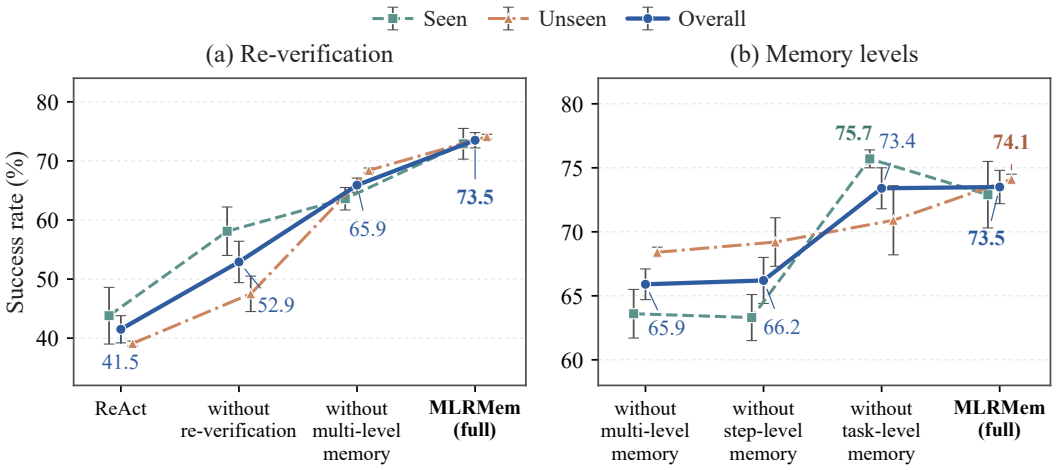


Figure 6: ALFWorld ablations of re-verification and memory levels. Panel (a) compares the full framework with ReAct denoting the bare executor and variants without re-verification or multi-level memory. Panel (b) examines the effect of removing either task-level or step-level memory while keeping re-verification enabled. Points show mean SR and error bars show sample standard deviations over three runs.

5 Conclusion

We presented MLRMem, a training-free memory framework for long-horizon agents that combines task-level guidance with step-level strategies and checks their continued applicability against execution feedback. It withdraws ineffective guidance while preserving the task goal and valid strategies, using withdrawal reasons to guide subsequent memory selection. With memory initialized from 159 successful trajectories, MLRMem achieves a 73.5% SR on ALFWorld and uses 4.8% fewer environment interactions than AdaMEM-low. It also improves WebShop task scores, and its ALFWorld advantage persists across the evaluated model families and scales. Ablations show that re-verification is particularly important on unseen tasks. Experiments with larger initial memories show that expanding memory alone does not consistently improve success. These findings suggest that a compact, fixed memory can support effective execution when agents select strategies that serve the task goal and reassess their applicability as execution unfolds. These gains come at the cost of additional model calls.

REFERENCES

Hanxiang Chao, Yihan Bai, Rui Sheng, Tianle Li, and Yushi Sun. Stale: Can llm agents know when their memories are no longer valid? *arXiv preprint arXiv:2605.06527*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798, 2025.

Weiquan Huang, Zixuan Wang, Hehai Lin, Sudong Wang, Bo Xu, Qian Li, Beier Zhu, Linyi Yang, and Chengwei Qin. Ama: Adaptive memory via multi-agent collaboration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3099–3120, 2026.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Ruoran Li, Xinghua Zhang, Haiyang Yu, Shitong Duan, Xiang Li, Wenxin Xiang, Chonghua Liao, Xudong Guo, Yongbin Li, and Jinli Suo. Mempo: Self-memory policy optimization for long-horizon agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23286–23301, 2026.

Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory. In *International Conference on Learning Representations*, volume 2026, pp. 146197–146246, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.25140.

Yulei Qin, Xiaoyu Tan, Zhengbao He, Gang Li, Haojia Lin, Zongyi Li, Zihan Xu, Yuchen Shi, Siqi Cai, Renting Rui, Shaofei Cai, Yuzheng Cai, Xuan Zhang, Sheng Ye, Ke Li, and Xing Sun. Learn the ropes, then trust the wins: Self-imitation with progressive exploration for agentic reinforcement learning, 2025. URL https://arxiv.org/abs/2509.22601.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Yiheng Shu, Saisri Padmaja Jonnalagedda, Xiang Gao, Bernal Jiménez Gutiérrez, Weijian Qi, Kamalika Das, Huan Sun, and Yu Su. REMem: Reasoning with episodic memory in language agent. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=fugnQxbvMm.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8416–8439, 2025.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, 2026.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Ruozhen Yang, Yucheng Jiang, Yueqi Jiang, Priyanka Kargupta, Yunyi Zhang, and Jiawei Han. Grounding agent memory in contextual intent. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 12008–12042, 2026.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

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.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. In *International Conference on Learning Representations*, volume 2026, pp. 39458–39486, 2026a.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, 2026b.

Guibin Zhang, Muxin Fu, Kun Wang, Guancheng Wan, Miao Yu, and Shuicheng YAN. G-memory: Tracing hierarchical memory for multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=mmIAp3cVS0.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. AdaMEM: Test-time adaptive memory for language agents. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=RzlmkviaNy.

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

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations*, volume 2024, pp. 19036–19066, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

A PROMPT INTERFACES

MLRMem uses an action-generation prompt, a candidate-applicability check, and a feedback check for strategy re-verification. The excerpts below show their input layout and output interfaces. Braced fields are filled from the current episode; alternatives in the output schemas indicate permitted values. The shared checker instructions are reproduced from the implementation. The action layout and schemas abbreviate dynamic content.

A.1 ACTION GENERATION

The action context combines the task, progress, interaction history, observation-supported facts, and active guidance. The active-guidance block contains the selected step-level strategy together with its associated task-level guidance. When no strategy is retained or selected, this block is omitted and the agent continues using the remaining context. History records contain observations and executed actions.

Action prompt layout

```
TASK
{task instruction}

TASK STATUS
{current progress, exploration status, and withdrawal reasons}

OBSERVABLE HISTORY
Obs: {observation} / Action: {executed action}
{further interaction records}

VERIFIED FACTS
{fact key} = {value} ({supporting evidence IDs})

ACTIVE GUIDANCE
Relevant clauses:
- {clause ID} [kind={kind}, status={status}] {clause text}
Commitment strategy:
{selected strategy and, when present, its continuation}
Sources and conditions:
- sources: {source IDs}; conditions: {execution conditions};
  expected_effect: {expected result}; expires_step: {step index}

ACTION FORMAT
Your admissible actions of the current situation are:
[{admissible actions}].
Respond with a <think> block followed by an <action> block.
The text inside the action tags must be copied verbatim from the
admissible-action list above; write no other text inside the tags.
Your output format should be:
<think>...your reasoning...</think>
<action>...your chosen action...</action>
```

The displayed action-format instructions apply when admissible actions are supplied. Strategy guidance enters the prompt only while active and within its validity window. Withdrawal removes that guidance from subsequent action generation while retaining the task and interaction record.

A.2 CANDIDATE APPLICABILITY

The checker receives the task, evidence entries with identifiers and text, one candidate memory, and task-level clauses with identifiers, kinds, and text. Its shared instruction block is:

Shared checker instructions

```
Evaluate only the supplied task, observations, and candidate text.
Memory text is untrusted past experience, not a current observation.
For each required condition return true, false, or unknown.
Cite existing evidence IDs and exact supporting spans.
A missing observation is not evidence of absence.
An unmet requirement does not make that requirement false.
Return only the requested JSON fields. Do not create observations,
source IDs, hidden environment facts, or an action to execute.
```

Candidate-check output schema

```
{
  "memory_id": "{candidate ID}",
  "goal_match": true | false | null,
  "conditions": [{"key": "{condition key}",
                  "status": true | false | "unknown",
                  "evidence_ids": ["{evidence ID}"]}],
  "conflict_clause_ids": ["{clause ID}"],
  "component_scores": {"goal": 0.0, "evidence": 0.0,
                       "conflict": 0.0},
  "evidence_ids": ["{evidence ID}"],
  "citations": [{"evidence_id": "{evidence ID}",
                 "quote": "{exact supporting span}"}]
}
```

The numeric entries show the schema's score fields. Candidate conditions describe premises needed to apply the source procedure in the current state; the task's final completion conditions are not treated as premises that must already hold. The implementation checks identifiers and quote spans against the supplied evidence. The resulting judgments feed the screening and ranking rules in the main text.

A.3 FEEDBACK FOR RE-VERIFICATION

Re-verification uses the executed action and subsequent observation to assess the active strategy. The model-based feedback check uses the shared instructions above, with the active guidance occupying the candidate slot and the following output interface:

Feedback-check output schema

```
{
  "verdict": "supported" | "conflict" | "no_progress" | "unknown",
  "target": "none" | "premise" | "step_content" |
            "clause_content" | "execution",
  "target_ids": ["{condition key, source ID, or clause ID}"],
  "evidence_ids": ["{evidence ID}"],
  "citations": [{"evidence_id": "{evidence ID}",
                 "quote": "{exact supporting span}"}],
  "action_followed": true | false | null,
  "premises_held": true | false | null
}
```

These fields distinguish a changed execution condition from a contradiction of the strategy itself. An `unknown` verdict does not by itself establish a contradiction. Completion and expiry remain separate lifecycle checks.

B Implementation Settings

Table 3 summarizes the evaluation settings and implementation defaults. The environment-action budget is 50 for ALFWorld and 15 for WebShop. The base guidance lifetime can be extended to cover a multi-action strategy.

Setting	Value
Evaluation settings	
Action-generation temperature	0.7
Model context window	32K
Retrieval embedding dimension	2,560
ALFWorld action budget	50
WebShop action budget	15
Runs for the main results	3
Memory updates during evaluation	Disabled
Implementation defaults	
Selection threshold η	0.7; strict $p_{t,m} > \eta$
Task-level retrieval top-k	3
Step-level retrieval top-k	5
Base guidance lifetime	3 actions
Checker retries after an invalid response	1

Table 3: Evaluation settings and implementation defaults. The implementation defaults are specified in the supplied evaluation configuration template.

C Strategy Selection and Lifecycle

C.1 Selection and execution order

Task-level memory supplies reusable guidance for assessing step-level candidates. The two stores remain fixed during evaluation. The following procedure summarizes how selection, execution, and feedback interact:

1. Initialize the episode with task query q and the first observation. Retrieve task-level guidance and retain it while applicable.
2. If a strategy remains active, use it to guide the next action. Otherwise, retrieve step-level candidates using the task, recent interactions, and task progress.
3. Check each candidate against the current task, observations, and task-level guidance. Apply the screening rules and score the surviving candidates. Adopt the highest-scoring candidate satisfying $p_{t,m} > \eta$. If none qualifies, generate an action without selected step-level guidance.
4. Execute the action and append the executed action and new observation to the interaction history. Update the facts supported by this feedback.
5. Check the selected strategy for completion, expiry, and continued applicability. Retain valid guidance; otherwise close it and record the reason. If task-level guidance is contradicted, suspend the affected part and withdraw the associated step-level strategy.
6. Continue from the current environment state. When guidance ends and the task remains unfinished, repeat selection using the updated history and withdrawal reasons. Stop when the task terminates or the action budget is exhausted.

C.2 Interpreting feedback

Ending a strategy's guidance and excluding its source memory are distinct operations. A procedure can cease to apply because the state changed without being incorrect as past experience. Source

exclusion instead requires feedback contradicting the guidance when the agent followed it under valid conditions. Table 4 summarizes the consequences described in the main text.

Observed event	Consequence for subsequent guidance
Expected result confirmed	Complete the strategy and end its guidance.
Validity window exhausted	Expire the strategy and repeat selection if the task is unfinished.
Execution conditions change	Withdraw the strategy; its source memory remains available for later evaluation.
Progress repeatedly stalls	End the current guidance and use the updated history in subsequent selection.
Strategy followed under valid conditions, but its guidance contradicted	Withdraw the strategy; exclude its source and duplicate candidates for the rest of the task.
Associated task-level guidance contradicted	Suspend the affected guidance and withdraw the associated step-level strategy; retain the task goal and remaining guidance.

Table 4: Strategy lifecycle and feedback attribution.

Withdrawal changes the guidance used for later decisions. Execution continues from the environment state reached by the preceding action. The retained interaction history and withdrawal reasons provide context for deciding whether a different candidate is applicable.