G-DREAMER: LONG-TERM MEMORY EVOLUTION AND MANAGEMENT FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly rely on external memory to support inference-time adaptation over long-term interaction. However, accumulating more experience does not necessarily sustain memory utility. As interaction continues, useful experience can become buried in a growing memory store, while redundant, outdated, or incorrect knowledge remains available for retrieval. Consequently, memory may perform well during early accumulation yet degrade substantially in later stages, even when its overall performance remains competitive. We refer to this phenomenon as *long-term degradation*. Mitigating such degradation requires maintaining an effective memory state that continuously incorporates new evidence, remains compact enough for reliable retrieval, and revises knowledge as its validity changes. Moreover, while concrete cases and abstract patterns provide complementary forms of experience, effective compression requires preserving their provenance to avoid losing supporting evidence. We introduce **G-Dreamer**, a framework for long-term memory evolution and management for LLM agents. During wake, G-Dreamer continuously collects trajectory-grounded cases and their outcomes. In the background, sleep-time lifecycle management performs consolidation to extract reusable patterns, selective forgetting to retire redundant experience while preserving provenance, and reconsolidation to revise patterns when later evidence challenges them. Adaptive scheduling coordinates these operations according to the evolving memory state and lifecycle stage. Experiments on long-horizon interactive tasks show that G-Dreamer mitigates long-term degradation, achieving stronger performance with a smaller active memory state at little additional maintenance cost. Code: `anonymous/G-Dreamer`.

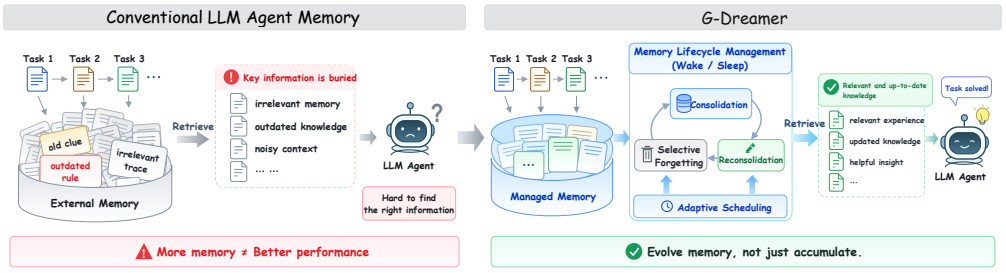

Figure 1: Motivation and overview of G-Dreamer. **(a)** Conventional memory suffers from **unmanaged accumulation** over long-term interaction. **(b)** G-Dreamer continually **reorganizes stored experience** according to its utility and newly observed evidence.

1 INTRODUCTION

Large language model (LLM) agents increasingly rely on external memory to carry useful experience across tasks and support inference-time adaptation without updating model parameters (Zhong et al., 2024; Shinn et al., 2023; Park et al., 2023; Suzgun et al., 2026; Qiu et al., 2026; Jin et al., 2025). Existing approaches store concrete interaction traces, extract reusable lessons or workflows, and organize experience through structured relations (Zhao et al., 2024; Wang et al., 2025; Ouyang et al., 2026; Xu et al., 2025; Jin et al., 2026; Fang et al., 2026a; Ye et al., 2026a).

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.

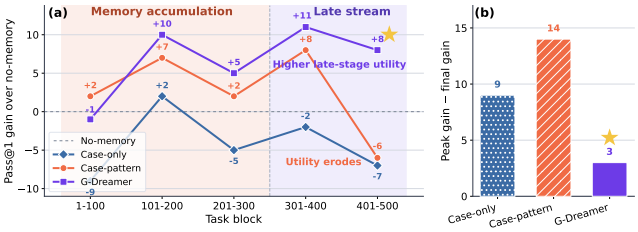

Figure 2: **Long-term degradation of memory utility** on ALFWorld tasks. **(a)** Case-only and case-pattern memory lose utility in the late stream, while G-Dreamer better preserves its gains. **(b)** Peak-to-final degradation is substantially smaller for G-Dreamer.

However, long-term interaction changes the objective: the challenge is no longer simply to accumulate experience, but to keep that experience useful as memory evolves. As Figure 1 illustrates, passive accumulation can leave relevant evidence buried among redundant or outdated records, whereas effective long-term memory requires continual management of what should remain accessible and up to date.

A central symptom is *long-term degradation*: memory may provide strong gains during early accumulation but lose much of that utility in later stages. Figure 2 shows this behavior directly: memory variants accumulating concrete experiences alone (case-only) or together with abstract patterns (case-pattern) reach useful intermediate states, but their late-stage gains fall sharply, whereas our method retains substantially more utility. Such degradation can be hidden by aggregate performance over the full task stream, even when the final memory state is markedly worse than an earlier one. As memory grows, the right experience may still be stored but become harder to retrieve under a limited context budget; meanwhile, stale or misleading records can continue to compete for retrieval. Prior studies similarly report sensitivity to irrelevant history, misleading retrieved experience, and continual-memory growth (Kim et al., 2026; Hu et al., 2026; Shao et al., 2026; Xiong et al., 2026). Thus, more memory does not necessarily imply more useful memory.

Mitigating this degradation requires maintaining an effective memory state rather than maximizing retention. New evidence should broaden memory coverage without allowing the active store to grow unchecked; later outcomes should also be able to revise knowledge that is no longer reliable. This creates a particular tension between *cases* and *patterns*. Concrete cases preserve task conditions and supporting evidence but accumulate rapidly; abstract patterns are easier to reuse, yet compression can remove the context needed to judge when a pattern applies (Zhao et al., 2024; Ouyang et al., 2026; Zhang et al., 2026b). Procedural knowledge can therefore indicate what to do yet still be misapplied when the relevant entities or task conditions are not preserved. Effective memory must guide both *what* procedure to follow and *where* its conditions apply. Repeated consolidation can also distort otherwise useful memories (Zhang et al., 2026a), while stale-memory studies highlight the need to revise knowledge rather than simply retrieve newer evidence (Chao et al., 2026). A long-term memory system should therefore preserve provenance links between cases and patterns, so that compression reduces redundancy without severing the evidence needed for later reassessment.

To this end, we introduce **G-Dreamer**, a lifecycle-based memory framework designed to preserve memory utility throughout long-term interaction. G-Dreamer adopts a wake–sleep organization inspired by complementary learning systems and biological memory maintenance (McClelland et al., 1995; Kumaran et al., 2016; Brodt et al., 2023; Nader, 2015; Richards & Frankland, 2017). During wake, the agent retains trajectory-level evidence that can later support or challenge higher-level abstractions. Sleep serves as ongoing maintenance rather than a post-hoc processing stage, targeting the main sources of long-term degradation. Recurrent successful or repaired experience is consolidated into reusable patterns, with provenance preserved to their supporting cases. As patterns stabilize, redundant cases are retired into semantic capsule anchors that retain key information and representative exemplars, reducing the active memory size without losing their evidential links. When later evidence challenges an established pattern, it is revised to reflect its updated validity or scope. Adaptive scheduling adjusts these maintenance operations to the maturity and current state of memory. This design keeps the active memory compact for retrieval while preserving the evidence required for reliable reuse and revision.

Our main contributions are:

(1) We propose a long-term memory management framework G-Dreamer that mitigates utility erosion through coordinated lifecycle operations of consolidation, forgetting, and reconsolidation.

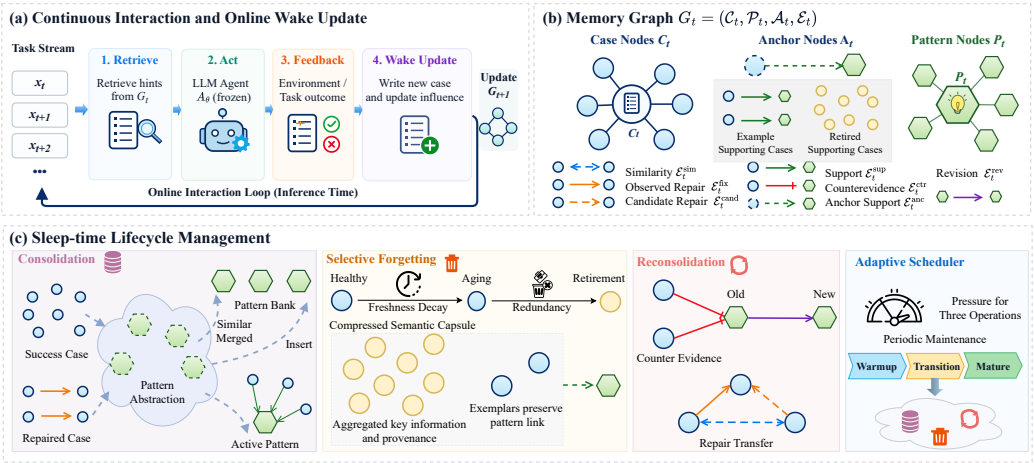

Figure 3: **The G-Dreamer framework.** **(a)** Online interaction continuously updates memory. **(b)** Cases, patterns, and anchors form the evolving memory graph. **(c)** Lifecycle operations and adaptive scheduling maintain long-term memory.

(2) We introduce a provenance-aware compression mechanism for case–pattern memory architectures that preserves representative evidence and traceability to reusable patterns while substantially reducing memory size.

(3) Experiments show that G-Dreamer sustains strong performance across benchmarks, achieving up to an 8.2% improvement in success rate while reducing the active memory size by over 50% compared with most baselines, demonstrating a favorable performance–efficiency trade-off. Long-term analysis further shows that G-Dreamer largely eliminates the utility degradation observed in accumulative memory systems.

2 G-DREAMER

Figure 3 depicts how G-Dreamer manages long-term memory within an evolving heterogeneous graph $G_t = (\mathcal{C}_t, \mathcal{P}_t, \mathcal{A}_t, \mathcal{E}_t)$. Concrete experience gathered during *wake* is gradually assimilated during *sleep*. During early accumulation, consolidation derives general insights from individual cases and records their provenance. As an insight gains sufficient support, retaining every source case becomes unnecessary; forgetting trims redundant records while anchors retain representative evidence. New experience may instead expose limitations of an existing insight, leading reconsolidation to reduce reliance on it and, when warranted, revise it. As memory grows, adaptive scheduling gradually shifts the emphasis from periodic organization of accumulated experience toward targeted maintenance, invoking the relevant operation as its need arises to support stable long-term reuse.

2.1 MEMORY GRAPH

Case Nodes and Relations. A case $c_i = (x_i, \tau_i, o_i, \mathbf{m}_i) \in \mathcal{C}_t$ records a concrete interaction through its task context, execution trajectory, outcome, and reuse statistics. According to o_i, cases are divided into successful $\mathcal{C}_t^+$ and failed $\mathcal{C}_t^-$ experiences. Their basic relations are

$$\mathcal{E}_t^{\text{sim}} = \{(c_i, c_j) \in \mathcal{C}_t^2 : s(c_i, c_j) \geq \tau_{\text{sim}}\}, \qquad \mathcal{E}_t^{\text{fix}} \subseteq \mathcal{C}_t^- \times \mathcal{C}_t^+, \tag{1}$$

where $s(c_i, c_j)$ measures case similarity and τ_{sim} is the minimum similarity threshold for establishing a case relation. $\mathcal{E}_t^{\text{fix}}$ records observed failure–repair pairs.

Pattern Nodes and Relations. A pattern $p_j = (\pi_j, \mathbf{q}_j) \in \mathcal{P}_t$ stores reusable knowledge π_j together with evolving properties $\mathbf{q}_j$, such as confidence, utility, and lifecycle state. Cases provide supporting and counterevidence through $\mathcal{E}_t^{\text{sup}}, \mathcal{E}_t^{\text{ctr}} \subseteq \mathcal{C}_t \times \mathcal{P}_t$. Pattern revisions are represented by $\mathcal{E}_t^{\text{rev}} \subseteq \mathcal{P}_t \times \mathcal{P}_t$, where $(p'_j, p_j) \in \mathcal{E}_t^{\text{rev}}$ indicates that p'_j supersedes its predecessor p_j.

Anchor nodes. When a stable pattern p_j is supported by a sufficient number of redundant cases, part of its supporting evidence can be retired into an anchor. Let $\mathcal{S}_t(p_j) = \{c_i : (c_i, p_j) \in \mathcal{E}_t^{\text{sup}}\}$. For $\mathcal{R}_k \subseteq \mathcal{S}_t(p_j)$, we define

$$a_k = (\mathcal{R}_k, \mathcal{X}_k, \bar{\mathbf{m}}_k) \in \mathcal{A}_t, \qquad \mathcal{X}_k \subseteq \mathcal{R}_k, \qquad (a_k, p_j) \in \mathcal{E}_t^{\text{anc}}, \tag{2}$$

where $\mathcal{X}_k$ retains a small set of exemplar cases for retrieval and $\bar{\mathbf{m}}_k$ aggregates statistics of the retired group. Anchors compress redundant evidence while preserving representative cases and their connection to the backing pattern.

2.2 Consolidation: From Experience to Reusable Patterns

Online task interactions shown in Figure 3 (a) generate concrete experiences that are promptly recorded in the case node. These records preserve task-specific solutions and failures, but relying on individual cases alone can limit the transfer beyond their original contexts. Consolidation therefore distills general procedures and corrective lessons from the collected cases, turning concrete experience into transferable patterns while preserving their provenance.

Pattern extraction. Successful cases provide solution traces, while failure–repair pairs capture corrections to failed executions. G-Dreamer extracts patterns from $c_i \in \mathcal{C}_t^+$ or $(c_f, c_r) \in \mathcal{E}_t^{\text{fix}}$:

$$\pi_i^{\text{succ}} = \Phi_{\text{s}}(c_i), \qquad \pi_{f,r}^{\text{rep}} = \Phi_{\text{r}}(c_f, c_r, \Delta(\tau_f, \tau_r)), \tag{3}$$

Here $\Delta(\tau_f, \tau_r)$ identifies the behavioral correction. The extraction functions remove source-specific bindings while preserving transferable procedures, so patterns guide related tasks rather than replaying a source trajectory verbatim.

Evidence accumulation and activation. Each extracted rule starts as a candidate linked to its source cases through $\mathcal{E}_t^{\text{sup}}$. It is merged with the most similar compatible pattern when their similarity meets τ_{merge}, or inserted otherwise, preserving provenance while limiting near-duplicate rules. With supporting cases $\mathcal{S}_t(p_j) = \{c_i : (c_i, p_j) \in \mathcal{E}_t^{\text{sup}}\}$, the pattern remains a candidate until its support and confidence satisfy

$$\text{Active}(p_j) = \mathbf{1}\left[N_{\text{ind}}(\mathcal{S}_t(p_j)) \geq \kappa_{\text{sup}} \ \wedge \ q_j^{\text{conf}} \geq \tau_{\text{act}}\right], \tag{4}$$

where N_{ind} counts independent sources rather than repeated observations of the same experience, q_j^{conf} denotes confidence, and κ_{sup} and τ_{act} are the support and confidence thresholds. Continued reuse updates pattern properties, allowing their influence to increase or decrease with new evidence. These evidence links support both compression of redundant cases and later revision.

2.3 Selective Forgetting: Retirement and Compression

As experience accumulates and patterns are consolidated, the memory graph continues to expand. Earlier experiences may become redundant or outdated over time, and a crowded memory can obscure the experiences that remain useful. To alleviate retrieval burden and prevent uncontrolled memory growth, targeted forgetting becomes necessary for preserving long-term memory utility.

Aging and case retirement. Cases that no longer warrant independent retention are first identified from their freshness and reuse history. For each case c_i, its freshness $F_t(c_i)$ is updated as

$$F_{t+1}(c_i) = \text{clip}\Big(F_t(c_i)e^{-\lambda_t(c_i)} + \delta_t^+(c_i) - \delta_t^-(c_i)\Big), \tag{5}$$

where $\lambda_t(c_i)$ is the case-specific decay rate, and $\delta_t^+(c_i)$ and $\delta_t^-(c_i)$ capture positive and adverse reuse signals, respectively. Cases with sustained utility therefore decay more slowly.

A retention score $\rho_t(c_i)$ then summarizes the accumulated value and redundancy of each case. Cases eligible for retirement are collected as

$$\mathcal{F}_t = \{c_i \in \mathcal{C}_t : \rho_t(c_i) < \tau_F \ \wedge \ c_i \notin \mathcal{K}_t\}, \tag{6}$$

where τ_F is the retirement threshold and $\mathcal{K}_t$ denotes cases protected by validated repair evidence. Thus, aging reflects how strongly a case should persist, while retirement determines whether it still needs to remain an independent active node.

Semantic capsule compression. Simply removing retired cases would discard the concrete evidence supporting their patterns. We instead introduce a *semantic capsule* that packages redundant supporting cases into an anchor node. As illustrated in Figure 3 (b), for a stable pattern p_j, a coherent retired group $\mathcal{R}_k \subseteq \mathcal{F}_t \cap \mathcal{S}_t(p_j)$ is compressed as

$$a_k = (\mathcal{R}_k, \mathcal{X}_k, \bar{\mathbf{m}}_k), \qquad \mathcal{X}_k = \mathrm{Exemplar}_2(\mathcal{R}_k), \qquad (a_k, p_j) \in \mathcal{E}_t^{\mathrm{anc}}. \tag{7}$$

The capsule acts as a container rather than a retrievable memory item: a_k preserves provenance, aggregate statistics, and the support relation to p_j, while the two representative cases in $\mathcal{X}_k$ remain available for ordinary retrieval as concrete supporting experience. The other redundant members no longer occupy separate active nodes or maintain repeated case–pattern connections.

This compression preserves both levels of memory: the pattern retains reusable knowledge, while the capsule keeps its concrete evidence traceable. By collapsing redundant cases and edges, it reduces graph growth and retrieval competition without severing the case–pattern relation, allowing a limited retrieval budget to focus more effectively on the memories that remain useful.

2.4 Reconsolidation: Revising Evolving Knowledge

Consolidated patterns are not always supported by later experience. New cases may contradict an existing pattern or expose limits to its applicability, making previously useful guidance less reliable. As counterevidence accumulates, the pattern needs to be reconsidered and revised so that reusable knowledge can evolve with continued experience rather than misguide future tasks. Reconsolidation addresses this pattern-level update. At the case level, it also revisits sparse but useful repair relations, allowing corrective experience to transfer from observed failures to compatible cases.

Pattern revision. Counterevidence $\mathcal{D}_t(p_j) = \{c_i : (c_i, p_j) \in \mathcal{E}_t^{\mathrm{ctr}}\}$ first lowers a pattern's confidence and retrieval priority. Sufficiently strong and diverse counterevidence then permits a scoped revision:

$$p_j' = \mathrm{ScopeRevise}(p_j, \mathcal{D}_t(p_j)), \qquad (p_j', p_j) \in \mathcal{E}_t^{\mathrm{rev}}. \tag{8}$$

The revision remains a candidate under revalidation, while its predecessor stays temporarily available to avoid overreacting to isolated contradictions.

Repair transfer. To extend corrections beyond their source failures, an observed repair (c_f, c_r) can propagate to a related failed case $c' \in \mathcal{C}_t^-$:

$$(c_f, c_r) \in \mathcal{E}_t^{\mathrm{fix}}, \quad (c', c_f) \in \mathcal{E}_t^{\mathrm{sim}}, \quad \chi(c', c_f, c_r) = 1 \Longrightarrow (c', c_r) \in \mathcal{E}_t^{\mathrm{cand}}. \tag{9}$$

Here χ enforces compatibility. Subsequent use validates or revokes candidate repairs, rather than treating similarity as proof of transferability.

2.5 Adaptive Maintenance Scheduling

Memory maintenance needs change as the graph matures. At the early stage, frequent sleep cycles help assimilate newly accumulated cases and establish reusable patterns. As memory grows, the benefit of repeatedly extracting more patterns diminishes, while maintaining existing knowledge through revision, retirement, and compression becomes increasingly important. A fixed sleep schedule therefore cannot reflect the changing demands of different maintenance operations. G-Dreamer combines stage-aware periodic maintenance with operation-specific scheduling. The memory stage $\sigma_t \in \{\mathrm{warmup}, \mathrm{transition}, \mathrm{mature}\}$ determines the periodic interval T_{σ_t}, while each operation $o \in \{\mathrm{C}, \mathrm{R}, \mathrm{F}\}$ has a pressure $P_{o,t} \in [0, 1]$ that measures its current urgency against threshold η_o. The selected set is

$$\mathcal{O}_t = \mathcal{O}_t^{\mathrm{per}}(\sigma_t) \cup \mathcal{O}_t^{\mathrm{prs}}, \qquad \mathcal{O}_t^{\mathrm{prs}} = \{\, o : P_{o,t} \geq \eta_o \,\}, \tag{10}$$

subject to operation-specific cooldown and hysteresis. This allows maintenance to shift from frequent organization during early accumulation toward targeted interventions as long-term memory becomes more mature. Further implementation details are provided in Appendix A.

Table 1: **Main results on ALFWorld and WebShop.** ALFWorld reports per-type success rates and their macro-average (SR, %); WebShop reports score (0–100). #Tok. denotes mean total LLM tokens per task (in thousands). Best results are shown in **bold**, and the second results are underlined.

		ALFWorld								WebShop	
Model	**Method**	**Pick**	**Look**	**Clean**	**Heat**	**Cool**	**Pick2**	**SR↑**	**#Tok.↓**	**Score↑**	**#Tok.↓**
Qwen3-8B	No Memory	43.4	41.8	55.4	58.8	39.8	70.8	51.7	3.13	23.5	16.4
	Case-Centric Accumulation										
	Case-only	47.4	60.0	64.4	61.2	48.2	79.2	60.1	3.67	22.3	20.2
	A-MEM	**61.6**	69.8	79.8	64.5	51.6	**87.2**	69.1	12.3	18.3	36.9
	Abstraction-Augmented Accumulation										
	Case-pattern	58.6	**81.2**	**85.8**	63.4	**73.2**	83.4	**74.3**	3.97	22.4	21.9
	ReasoningBank	43.3	54.0	66.0	58.8	22.6	81.8	54.4	5.74	20.2	23.0
	ExpeL	51.0	62.6	61.4	55.4	46.0	84.0	60.1	3.82	24.1	17.6
	Lifecycle-Managed Memory										
	G-Dreamer	59.8	71.2	78.2	**76.8**	64.0	86.4	72.7	4.09	**25.1**	20.5
Qwen3-14B	No Memory	52.2	52.2	69.4	70.8	46.4	81.2	62.0	3.60	37.7	7.48
	Case-Centric Accumulation										
	Case-only	41.2	54.6	70.6	66.6	43.2	86.4	60.4	3.53	32.1	13.1
	A-MEM	**62.8**	**74.0**	81.6	**83.2**	61.4	87.4	**75.1**	11.4	31.9	19.8
	Abstraction-Augmented Accumulation										
	Case-pattern	44.2	69.4	79.6	73.4	**70.0**	87.6	70.7	3.84	42.1	13.2
	ReasoningBank	58.9	59.8	73.4	71.6	43.6	81.8	64.9	4.90	**53.0**	11.7
	ExpeL	56.8	49.4	80.2	72.8	50.2	87.6	66.2	4.28	37.4	11.2
	Lifecycle-Managed Memory										
	G-Dreamer	43.4	68.4	**82.6**	79.4	65.4	**92.0**	71.9	3.73	44.7	12.7

3 EXPERIMENTS

We evaluate G-Dreamer through four research questions: **RQ1:** Does G-Dreamer sustain stronger performance over long-term task streams? **RQ2:** What performance–efficiency trade-off does it achieve as memory evolves? **RQ3:** How do consolidation and lifecycle management affect long-term performance and transfer? **RQ4:** How does the memory guide task execution in practice?

Datasets and Baselines We evaluate on selected task subsets from **ALFWorld** (Shridhar et al., 2021) and **WebShop** (Yao et al., 2022), two interactive benchmarks requiring multi-step decision making. We compare against two common forms of existing memory systems. *Case-centric accumulation* retains concrete, instance-level experience from previous interactions, represented by **Case-only**, a case-graph adaptated from EXG (Jin et al., 2026), and **A-MEM** (Xu et al., 2025), which organizes individual memories through dynamic indexing and linking. *Abstraction-augmented accumulation* further extracts reusable higher-level knowledge beyond individual cases. This group includes **Case-pattern**, which augments the same case backbone with consolidated patterns, **ReasoningBank** (Ouyang et al., 2026), which distills generalizable reasoning strategies, and **ExpeL** (Zhao et al., 2024), which derives reusable insights from prior trajectories. Although Case-only and Case-pattern also serve as controlled variants of G-Dreamer, they instantiate these two representative memory paradigms. In contrast, **G-Dreamer** moves beyond accumulation to *lifecycle-managed memory*, continually maintaining case–pattern memory through selective forgetting, reconsolidation, and adaptive maintenance. All experiments use frozen models from the **Qwen3** family (Qwen Team, 2025), specifically Qwen3-8B and Qwen3-14B. Dataset subsets, baseline adaptations, and implementation details are provided in Appendix B.

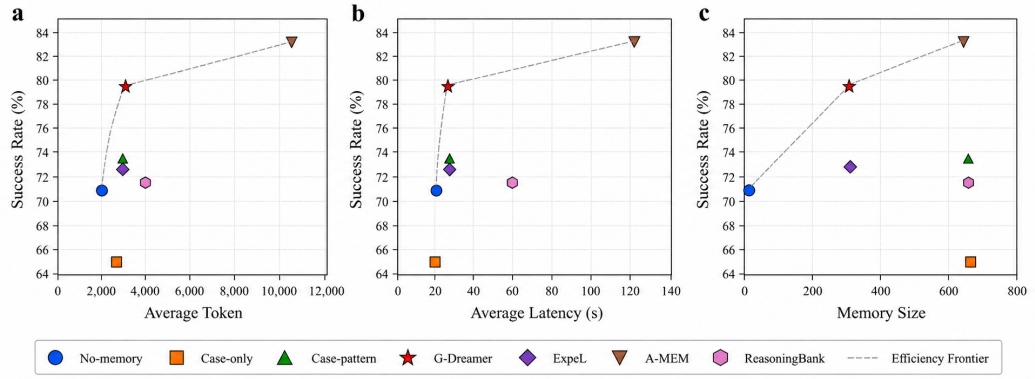

Figure 4: **Trade-off between performance and efficiency on ALFWorld.** We compare different memory methods in terms of (a) average tokens per task, (b) average latency per task, and (c) memory size (number of nodes or items). Our method achieves higher success rates than existing memory baseline with competitive computational cost and memory size.

3.1 Main Results (RQ1)

Table 1 answers **RQ1**: G-Dreamer achieves the best or second-best result in most model–task settings. On the six ALFWorld task families, G-Dreamer shows consistent gains across both model sizes. With Qwen3-8B, the largest improvement over No Memory occurs on `Look`, where the success rate increases from 41.8% to 71.2% (+29.4 percentage points). On `Heat`, G-Dreamer reaches 76.8%, outperforming the strongest competing baseline, A-MEM, by 12.3 points (64.5%), which is its largest margin over existing memory methods. With Qwen3-14B, G-Dreamer further reaches 92.0% on `Pick2`, the highest absolute success rate among all ALFWorld results.

We next consider the average performance across all six ALFWorld task families. With Qwen3-8B, our method variant Case-pattern achieves the highest macro-average SR of 74.3%, followed by G-Dreamer at 72.7%; with Qwen3-14B, A-MEM ranks first at 75.1%, while G-Dreamer remains second at 71.9%. For the smaller model, retaining a larger pool of accumulated memory may be helpful when model capacity is limited. Nevertheless, after memory compression, G-Dreamer preserves 97.8% of the Case-pattern performance. The strong Case-pattern result also supports the effectiveness of consolidation. For the larger model, A-MEM achieves a slightly higher SR through substantially more model calls to refine its memory, but requires 11.4K tokens per task, over three times the 3.73K tokens used by G-Dreamer. G-Dreamer therefore achieves a comparable SR while reducing average token cost by 67.3%, showing a better performance–computation trade-off.

On WebShop, G-Dreamer ranks first with Qwen3-8B (25.1) and second with Qwen3-14B (44.7). Across model scales, pattern-based methods achieve stronger performance than methods relying mainly on individual cases. Compared with accumulative pattern memory, G-Dreamer further controls memory growth through forgetting and continuously revises patterns as new evidence arrives. By reducing redundant or outdated memory while correcting accumulated patterns, G-Dreamer increases the chance of retrieving useful patterns and thereby achieves better overall results.

3.2 Efficiency (RQ2)

Figure 4 answers **RQ2** by examining the performance-efficiency trade-off from token consumption, latency, and long-term memory size. Across all three dimensions, G-Dreamer lies on the efficiency frontier, where each point represents a non-dominated operating point for which higher performance can only be obtained at greater resource cost.

In Figures 4 (a) and (b), accumulation-based methods largely cluster in the low-cost region; compared with the strongest method in this region, Case-pattern, G-Dreamer improves success rate by approximately 8.2% while requiring comparable token cost and slightly lower latency. A-MEM attains the highest success rate, but G-Dreamer remains within roughly 4.6% of its performance while using only about 28.8% of its tokens and 20.5% of its latency. A similar pattern holds for the long-

term memory state in Figure 4 (c). At a memory size comparable to ExpeL, G-Dreamer achieves approximately 9.1% higher success rate, whereas several substantially larger memory states yield lower performance. Compared with A-MEM, G-Dreamer maintains a roughly 53.1% smaller active memory state while remaining close in task performance. For structured memories, the resulting graph is also substantially sparser, reducing the number of edges by 53.6% relative to the largest graph. Overall, G-Dreamer achieves stronger memory utility without proportionally increasing either inference cost or the size of the maintained memory state.

3.3 Ablation and Transfer (RQ3)

Table 2 evaluates the two successive design stages of G-Dreamer. To distinguish beneficial transfer from regressions across the same tasks, **PTR** (Positive Transfer Rate) captures failure-to-success transitions, **NTR** (Negative Transfer Rate) captures success-to-failure transitions, and **FTG** (Forward Transfer Gain) reflects the overall net transfer gain. Adding patterns and consolidation to Case-only yields a 6.8-point FTG, while adding the remaining lifecycle management to Case-pattern provides a further 6.0-point gain. In both stages, PTR remains substantially higher than NTR, indicating that each design stage produces a positive net contribution rather than merely trading improvements on some tasks for regressions on others.

Table 2: **Pairwise transfer analysis on ALFWorld (Heat).** The stepwise gains, together with higher PTR and lower NTR, show that each added component yields genuinely beneficial transfer.

Comparison	FTG ↑	PTR ↑	NTR ↓
Case-only → Case-pattern	+6.8%	18.0%	11.2%
Case-pattern → G-Dreamer	+6.0%	16.0%	10.0%
Case-only → G-Dreamer	+12.8%	21.6%	8.8%

To examine the effect of lifecycle management more directly, we further use Case-only as a common reference. Compared with Case-pattern, full G-Dreamer raises PTR from 18.0% to 21.6% while reducing NTR from 11.2% to 8.8%. The resulting 6.0-point additional FTG therefore consists of 3.6 points more failure-to-success transitions together with 2.4 points fewer success-to-failure transitions. This suggests that long-term memory management improves reuse in two complementary ways: it enables accumulated experience to solve more tasks that were previously unsuccessful, while reducing interference with tasks that were already solved correctly.

3.4 Case Study (RQ4)

Figure 5 examines a Heat task that requires retrieving a bowl, microwaving it, and placing it on cabinet 1. Case-only retrieves successful experience mainly as abstract phase sequences, which provides the overall procedure but insufficient information for grounding the bowl's source; the agent therefore proceeds to the microwave without first acquiring the correct object. Case-pattern further retrieves reusable heat-and-place rules and interaction preconditions, yet still binds the bowl to countertop 1 rather than cabinet 1. Thus, abstraction improves procedural guidance, but does not by itself ensure that the procedure is applied to the correct entities and conditions.

G-Dreamer retrieves a more discriminative combination of case evidence and repair-oriented guidance. This case illustrates how preserving provenance and traceability between patterns and their supporting cases helps ground reusable procedures in the concrete conditions under which they should be applied. Its selected cases retain source-side structural information, while the retrieved repair knowledge emphasizes opening the source container before taking the object. This changes the critical first step to *open cabinet 1* → *take bowl*, after which the remaining heating and delivery steps are completed correctly in 14 steps. This answers **RQ4**: effective memory must guide both *what* procedure to follow and *where* its conditions apply.

4 Related Work

Experience Memory. Experience-based memory has evolved from retaining individual episodes to constructing reusable abstractions. Case-based methods preserve trajectories, reflections, or successful demonstrations for later reuse (Park et al., 2023; Shinn et al., 2023; Zheng et al., 2024; Zhao et al., 2024). Pattern-oriented approaches distill recurring experience into workflows, strategies, or skills (Wang et al., 2025; Ouyang et al., 2026; Fang et al., 2026b; Forouzandeh et al., 2026; Yang

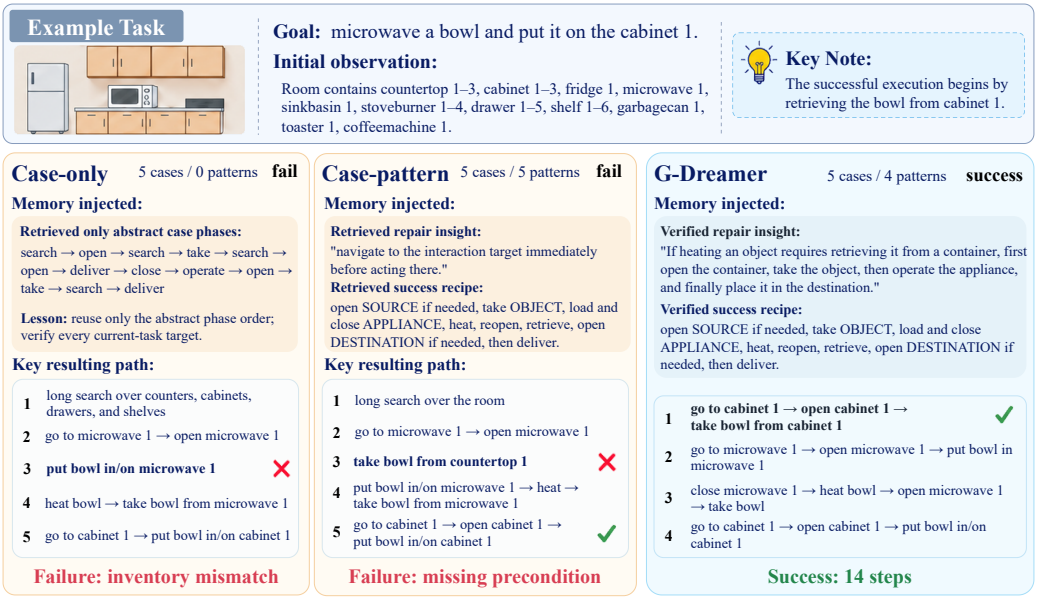

Figure 5: **Case study on an ALFWorld Heat task.** Case-only provides a coarse routine, Case-pattern adds procedural guidance but mis-grounds the source, while G-Dreamer retrieves grounded case and repair information for successful execution.

et al., 2026; Lin et al., 2026). Structured methods further organize experience through graphs, or abstraction hierarchies (Xu et al., 2025; Jin et al., 2026; Tan et al., 2026; Liang et al., 2026). Experience can also be refined through offline extraction or evolved online with execution feedback (Suzgun et al., 2026; Zhang et al., 2026b;d;c; Feng et al., 2026; Ye et al., 2026b; Cao et al., 2026). Despite these advances, most methods still center on accumulating and reusing experience, leaving sustained memory-state maintenance less explored (Xiong et al., 2026).

Long-Term Memory. Long-term memory instead concerns whether stored experience remains usable over time. Recent systems move beyond retention toward continual organization and adaptation, allowing accumulated memory to support persistent and cross-task use (Zhong et al., 2024; Gutiérrez et al., 2024; Qiu et al., 2026; Chhikara et al., 2025; Yu et al., 2026; Ke et al., 2026). Another line studies how memory quality degrades under accumulation (Zhang et al., 2026a) and introduces selective mechanisms to suppress redundant information (Wei et al., 2026; Chao et al., 2026; Kim et al., 2026; Shao et al., 2026; Fan et al., 2026; Hu et al., 2026). Complementary work explores periodic or offline consolidation, sometimes inspired by sleep, to reorganize accumulated memory (Fang et al., 2026a; Ye et al., 2026a; Dai et al., 2026; Xie, 2026; Lee et al., 2026). These efforts typically optimize particular operations or timescales rather than the memory lifecycle as a whole, G-Dreamer focuses on maintaining an effective memory state with coordinated control.

5 Conclusion

This paper presents G-Dreamer, a management framework for LLM agent memory that targets long-term degradation under sustained experience growth, where useful evidence becomes harder to access. Consolidation abstracts successful and repaired trajectories into reusable patterns, selective forgetting reduces retrieval interference while retaining provenance links, and reconsolidation revises patterns when new evidence challenges their validity or scope. Experiments on ALFWorld and WebShop show that G-Dreamer sustains strong performance across benchmarks, outperforming the strongest comparable-cost accumulation baseline while maintaining over 50% smaller active memory. Long-term and case analyses further show that G-Dreamer preserves memory utility by retaining reusable knowledge with its applicable context. These results suggest that maintaining an effective, evolving memory state, rather than continually accumulating experience, is a promising direction for scalable long-term agent memory.

AI USE STATEMENT

We used generative AI tools to polish author-written text, assist with code writing, improve manuscript presentation, and identify relevant literature. All AI-assisted content was reviewed, revised, and verified by the authors. We take responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the implementation and experimental details necessary to reproduce our results in the appendix. We report the model backbones, benchmark splits, evaluation metrics, baseline implementations, and experimental protocols used in our experiments. Additional utility and efficiency analyses are included to support the main findings. Anonymous source code is also provided.

REFERENCES

Svenja Brodt, Marion Inostroza, Niels Niethard, and Jan Born. Sleep—a brain-state serving systems memory consolidation. *Neuron*, 111(7):1050–1075, 2023. doi: 10.1016/j.neuron.2023.03.005. URL https://doi.org/10.1016/j.neuron.2023.03.005.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.829. URL https://aclanthology.org/2026.findings-acl.829/.

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. URL https://arxiv.org/abs/2605.06527.

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. URL https://arxiv.org/abs/2504.19413.

Zijie Dai, Shiyuan Deng, Sheng Guan, Yizhou Tian, Xin Yao, Xiao Yan, and James Cheng. RecMem: Recurrence-based memory consolidation for efficient and effective long-running LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32353–32376, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1619. URL https://aclanthology.org/2026.findings-acl.1619/.

Linfeng Fan, Yuan Tian, Ziwei Li, and Zhiwu Lu. Skill drift is contract violation: Proactive maintenance for LLM agent skill libraries. arXiv preprint arXiv:2605.10990, 2026. URL https://arxiv.org/abs/2605.10990.

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 *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a05b72653ec5b473732129829ae04195-Abstract-Conference.html.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. MemP: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, San Diego, California, United States, 2026b. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.866. URL https://aclanthology.org/2026.findings-acl.866/.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, and Jiaxuan You. ExpGraph: Model-agnostic experience learning with graph-structured memory for LLM agents. arXiv preprint arXiv:2605.30712, 2026. URL https://arxiv.org/abs/2605.30712.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for LLM agents through bayesian selection and contrastive refinement. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1820–1828, Paphos, Cyprus, 2026. International Foundation for Autonomous Agents and Multiagent Systems. doi: 10.65109/FKYO8341. URL https://doi.org/10.65109/FKYO8341.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 59532–59569, 2024. doi: 10.52202/079017-1902. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/6ddc001d07ca4f319af96a3024f6dbd1-Abstract.html.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in LLM agents. arXiv preprint arXiv:2604.27003, 2026. URL https://arxiv.org/abs/2604.27003.

Yuxin Jin, Hanchen Wang, Ying Zhang, Dong Wen, Lu Qin, and Wenjie Zhang. Data in agentic AI: A comprehensive survey. *TechRxiv*, 2025. doi: 10.36227/techrxiv.176704339.93520117/v1. URL https://www.techrxiv.org/doi/full/10.36227/techrxiv.176704339.93520117/v1.

Yuxin Jin, Siyuan Zhang, Hanchen Wang, Lu Qin, Ying Zhang, and Wenjie Zhang. EXG: Self-evolving agents with experience graphs. arXiv preprint arXiv:2605.17721, 2026. URL https://arxiv.org/abs/2605.17721.

Cai Ke, Jiangyue Yan, Han Zhang, Xin Liu, Zike Yuan, Yue Yu, Hui Wang, and Ruifeng Xu. Interactive memory learning for long-term conversations. arXiv preprint arXiv:2609.17088, 2026. URL https://arxiv.org/abs/2609.17088.

Minjae Kim, Jinheon Baek, Soyeong Jeong, and Sung Ju Hwang. MemRefine: LLM-guided compression for long-term agent memory. arXiv preprint arXiv:2606.13177, 2026. URL https://arxiv.org/abs/2606.13177.

Dharshan Kumaran, Demis Hassabis, and James L. McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(7):512–534, 2016. doi: 10.1016/j.tics.2016.05.004. URL https://doi.org/10.1016/j.tics.2016.05.004.

Sangyun Lee, Sean McLeish, Tom Goldstein, and Giulia Fanti. Do language models need sleep? offline recurrence for improved online inference. arXiv preprint arXiv:2605.26099, 2026. URL https://arxiv.org/abs/2605.26099.

Sirui Liang, Pengfei Cao, Jian Zhao, Wenhao Teng, Xiangwen Liao, Jun Zhao, and Kang Liu. Learning how to remember: A meta-cognitive management method for structured and transferable agent memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30733–30753, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1535. URL https://aclanthology.org/2026.findings-acl.1535/.

Zihan Lin, Zhenyu Chen, Jiawen Wei, Xiaohan Wang, Jie Cao, Jiajun Chai, Wei Lin, Guojun Yin, and Ran He. When not to imitate: Boundary-aware skill memory for reliable tool-use LLM agents. arXiv preprint arXiv:2608.22339, 2026. URL https://arxiv.org/abs/2608.22339.

James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457, 1995. doi: 10.1037/0033-295X.102.3.419. URL https://doi.org/10.1037/0033-295X.102.3.419.

Karim Nader. Reconsolidation and the dynamic nature of memory. *Cold Spring Harbor Perspectives in Biology*, 7(10):a021782, 2015. doi: 10.1101/cshperspect.a021782. URL https://doi.org/10.1101/cshperspect.a021782.

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://openreview.net/forum?id=jL7fwchScm.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, pp. 1–22, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Yuli Qiu, Yutong Li, Wei Su, Zeming Liu, Wanxiang Che, Heyan Huang, Haifeng Wang, and Yuang Guo. LifeMem: Enabling lifelong experience reuse for LLM agents. arXiv preprint arXiv:2609.12655, 2026. URL https://arxiv.org/abs/2609.12655.

Qwen Team. Qwen3 Technical Report. arXiv preprint arXiv:2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Blake A. Richards and Paul W. Frankland. The persistence and transience of memory. *Neuron*, 94(6):1071–1084, 2017. doi: 10.1016/j.neuron.2017.04.037. URL https://doi.org/10.1016/j.neuron.2017.04.037.

Jiaqi Shao, Yiyi Lu, Yunzhen Zhang, and Bing Luo. When stored evidence stops being usable: Scale-conditioned evaluation of agent memory. arXiv preprint arXiv:2605.07313, 2026. URL https://arxiv.org/abs/2605.07313.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

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, Rabat, Morocco, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

Zhixun Tan, Qiang Chen, Tairan Huang, Xiu Su, and Yi Chen. ConMem: Structured memory-guided adaptation in training-free multi-agent systems. arXiv preprint arXiv:2606.08702, 2026. URL https://arxiv.org/abs/2606.08702.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Lei Wei, Xiao Peng, Xu Dong, Niantao Xie, and Bin Wang. FadeMem: Biologically-inspired forgetting for efficient agent memory. arXiv preprint arXiv:2601.18642, 2026. URL https://arxiv.org/abs/2601.18642.

Ying Xie. Learning to forget: Sleep-inspired memory consolidation for resolving proactive interference in large language models. arXiv preprint arXiv:2603.14517, 2026. URL https://arxiv.org/abs/2603.14517.

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. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.27. URL https://aclanthology.org/2026.acl-long.27/.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604, 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/19909c36f51abc4856b4560aff3d36d6-Paper-Conference.pdf.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. AutoSkill: Experience-driven lifelong learning via skill self-evolution. arXiv preprint arXiv:2603.01145, 2026. URL https://arxiv.org/abs/2603.01145.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022. doi: 10.52202/068431-1508.

Chongrui Ye, Yuxiang Liu, Yu Wang, Haofei Yu, Yining Zhao, Ge Liu, Julian McAuley, and Jiaxuan You. Auto-Dreamer: Learning offline memory consolidation for language agents. arXiv preprint arXiv:2605.20616, 2026a. URL https://arxiv.org/abs/2605.20616.

Yongshi Ye, Tian Lan, Feihu Jiang, Muyang Ye, Bin Zhu, Qianghuai Jia, Longyue Wang, Zhao Xu, Weihua Luo, and Xiaodong Shi. CHIME: Credit-aware hierarchical memory evolution for long-horizon agentic planning. arXiv preprint arXiv:2609.02074, 2026b. URL https://arxiv.org/abs/2609.02074.

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, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.981. URL https://aclanthology.org/2026.acl-long.981/.

Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by LLMs. arXiv preprint arXiv:2605.12978, 2026a. URL https://arxiv.org/abs/2605.12978.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

Yaolun Zhang, Yiran Wu, Yijiong Yu, Qingyun Wu, and Huazheng Wang. Live-Evo: Online evolution of agentic memory from continuous feedback. arXiv preprint arXiv:2602.02369, 2026c. URL https://arxiv.org/abs/2602.02369.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. AdaMEM: Test-time adaptive memory for language agents. arXiv preprint arXiv:2606.05684, 2026d. URL https://arxiv.org/abs/2606.05684.

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. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

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*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/52f050499cf82fa8efb588e263f6f3a7-Abstract-Conference.html.

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. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

A METHOD DETAILS

This appendix supplements Section 2 with the online update process, pattern abstraction and reuse evidence, selective forgetting, reconsolidation, and adaptive maintenance scheduling. It uses the memory graph and case relations defined in Section 2.1.

A.1 ONLINE INTERACTION AND MEMORY UPDATE

G-Dreamer maintains external memory for a frozen agent A_θ over a continual task stream. For each task x_t, the agent retrieves memory context $\mathcal{H}_t$ and produces output $\hat{y}_t$ and trajectory τ_t:

$$\mathcal{H}_t = \mathrm{Retrieve}(G_t, x_t), \qquad (\hat{y}_t, \tau_t) = A_\theta(x_t, \mathcal{H}_t). \tag{11}$$

The context provides concrete case experience and reusable pattern knowledge.

After feedback f_t becomes available, the interaction is encoded as $c_t = \mathrm{Encode}(x_t, \tau_t, f_t)$ and incorporated into the graph:

$$\widetilde{G}_t = \mathrm{WakeUpdate}(G_t, c_t, \mathcal{H}_t, f_t), \qquad G_{t+1} = \mathrm{Maintain}(\widetilde{G}_t; \mathcal{O}_t). \tag{12}$$

The wake update inserts the new experience, records its case relations, and updates the reuse statistics of retrieved memories. The resulting graph guides the next interaction without updating the underlying LLM parameters.

A.2 PATTERN ABSTRACTION AND REUSE EVIDENCE

Candidate merging. Each extracted content π initializes a candidate $p = (\pi, \mathbf{q})$, linked to its source cases through $\mathcal{E}_t^{\mathrm{sup}}$. Let $\mathcal{P}_t^{\mathrm{cmp}}(\pi) \subseteq \mathcal{P}_t$ denote the existing patterns compatible with π. If this set is empty, the candidate is inserted directly. Otherwise, the most similar compatible pattern is selected:

$$\begin{aligned} p^* &= \arg \max_{p_j \in \mathcal{P}_t^{\mathrm{cmp}}(\pi)} s_p(\pi, \pi_j), \\ p &\leftarrow \begin{cases} \mathrm{Merge}(p, p^*), & s_p(\pi, \pi^*) \geq \tau_{\mathrm{merge}}, \\ \mathrm{Insert}(p), & \text{otherwise}, \end{cases} \end{aligned} \tag{13}$$

where π^* is the content of p^*, s_p measures semantic similarity, and τ_{merge} controls deduplication. Merging retains a single reusable rule while accumulating its provenance and supporting evidence.

Reuse credit. Let $\mathcal{P}_t^h \subseteq \mathcal{P}_t$ be the patterns included in the memory context $\mathcal{H}_t$, and let $r_t(p_j)$ be the retrieval score of p_j. For a nonempty retrieved pattern set, credit is distributed as

$$w_t(p_j) = \frac{\max(\epsilon, r_t(p_j))}{\sum_{p \in \mathcal{P}_t^h} \max(\epsilon, r_t(p))}, \qquad \epsilon > 0. \tag{14}$$

No pattern credit is assigned when $\mathcal{P}_t^h$ is empty. For a graded outcome $g_t \in \{1, \alpha, -1\}$ representing strict success, partial success, and failure, respectively, accumulated pattern utility is updated by

$$U_{t+1}(p_j) = U_t(p_j) + \mathbf{1}[w_t(p_j) \geq \tau_u] w_t(p_j) g_t, \tag{15}$$

where α is the partial-success credit and τ_u filters negligible credit. Corresponding exposure, help, and harm statistics are stored in $\mathbf{q}_j$. Case-level counters in $\mathbf{m}_i$ are updated from the same observed outcome. These statistics measure association with subsequent task outcomes; they do not identify the causal contribution of an individual memory item.

A.3 CASE RETIREMENT AND SEMANTIC CAPSULE COMPRESSION

Importance-dependent aging. For a case c_i, let $U_t(c_i)$ denote accumulated reuse utility and $H_t(c_i), R_t(c_i)$ denote helpful reuse and retrieval counts. Its importance and decay rate are

$$\begin{aligned} I_t(c_i) &= \beta_U [U_t(c_i)]_+ + \beta_H H_t(c_i) + \beta_R R_t(c_i), \\ \lambda_t(c_i) &= \lambda_0 e^{-\gamma I_t(c_i)}. \end{aligned} \tag{16}$$

Here $[u]_+ = \max(0, u)$. Greater importance slows the freshness decay in Equation 5; aging alone does not remove a case.

Aging adjustments. The bounded terms $\delta_t^+(c_i)$ and $\delta_t^-(c_i)$ in Equation 5 summarize positive and adverse reuse since the previous aging update. A generic parameterization is

$$\begin{aligned}\delta_t^+(c_i) &= \min\big(\delta_{\max}^+, \alpha_H\,\Delta H_t(c_i) + \alpha_R\,\Delta R_t(c_i)\big)\,,\\ \delta_t^-(c_i) &= \min\big(\delta_{\max}^-, \alpha_D\,\Delta D_t(c_i)\big)\,,\end{aligned} \tag{17}$$

where ΔH_t, ΔR_t, and ΔD_t denote newly accumulated helpful reuse, retrieval exposure, and harmful reuse. The caps prevent a single update from dominating the freshness state.

Retention score and protection. The retention score summarizes a case's accumulated value and redundancy:

$$\rho_t(c_i) = \mathrm{RetainScore}(c_i, \mathbf{m}_i, G_t)\,, \tag{18}$$

which is instantiated as

$$\rho_t(c_i) = \mathbf{w}_+^\top \mathbf{z}_t^+(c_i) - \mathbf{w}_-^\top \mathbf{z}_t^-(c_i), \tag{19}$$

where $\mathbf{z}_t^+$ collects evidence for preserving a case (e.g., useful reuse, freshness, distinctiveness, and repair value) and $\mathbf{z}_t^-$ collects evidence for retirement (e.g., harmful reuse, staleness, interference, and redundancy). The weights are configuration parameters. The protection set $\mathcal{K}_t$ in Equation 6 contains cases required to preserve validated repair evidence, so a low retention score alone does not remove an established correction.

Near-duplicate merging. Before group-level compression, cases already selected for retirement may be merged when

$$s(c_i, c_j) \geq \tau_{\mathrm{dup}} \quad \wedge \quad \mathrm{Compat}(c_i, c_j) = 1, \tag{20}$$

where Compat enforces agreement on outcome and the experience-level attributes required by the task family. One representative case is retained and the reuse statistics of merged members are aggregated into it. This step reduces local redundancy before semantic compression.

Semantic capsule eligibility. Using the anchor representation in Equation 2, a retired supporting group $\mathcal{R}_k \subseteq \mathcal{F}_t \cap \mathcal{S}_t(p_j)$ can be compressed only when p_j provides a stable active backing pattern. The group must also contain sufficient evidence and be semantically coherent. A generic criterion is

$$|\mathcal{R}_k| \geq \kappa_A, \qquad \mathrm{Coh}(\mathcal{R}_k) = \frac{2}{|\mathcal{R}_k|(|\mathcal{R}_k|-1)} \sum_{\substack{c_i, c_j \in \mathcal{R}_k \\ i<j}} s(c_i, c_j) \geq \tau_A. \tag{21}$$

with configurable evidence threshold κ_A and coherence threshold τ_A.

Two-exemplar compression and retrieval. Each capsule retains two representative cases, as specified in Equation 7. Selection covers the retired group rather than merely favoring recent cases; one coverage-based realization is

$$\mathcal{X}_k = \arg\max_{\substack{\mathcal{X} \subseteq \mathcal{R}_k \\ |\mathcal{X}|=2}} \sum_{c \in \mathcal{R}_k} \max_{e \in \mathcal{X}} s(c, e). \tag{22}$$

The two cases in $\mathcal{X}_k$ remain available for ordinary case retrieval. Other redundant members no longer occupy separate active nodes or maintain repeated case–pattern connections. Instead, the anchor a_k preserves their provenance and aggregate statistics $\bar{\mathbf{m}}_k$, with $(a_k, p_j) \in \mathcal{E}_t^{\mathrm{anc}}$ retaining the backing relation. The anchor is a bookkeeping container, not another reusable rule or an independently retrievable memory item. Compression preserves recorded support rather than creating new evidence; an exemplar and its anchor do not count as independent sources of the same experience.

A.4 Reconsolidation and Candidate Repair Transfer

Counterevidence downweighting. For pattern p_j, let $S_{j,t}$ and $C_{j,t}$ denote its recorded support and counterevidence counts. For uncompressed evidence, these are $|\mathcal{S}_t(p_j)|$ and $|\mathcal{D}_t(p_j)|$, respectively; compressed support remains recorded through anchor provenance and aggregate statistics. Their ratio

$$\rho_{j,t}^{\mathrm{ctr}} = \frac{C_{j,t}}{\max(1, S_{j,t} + C_{j,t})} \tag{23}$$

is used together with reuse statistics to reduce confidence and retrieval priority before any text revision is attempted. For newly observed counterevidence $\Delta C_{j,t}$, a generic update is

$$q_{j,t+1}^{\mathrm{conf}} = \mathrm{clip}\big(q_{j,t}^{\mathrm{conf}} - \eta_{\mathrm{ctr}}\alpha_{j,t}\Delta C_{j,t}\big)\,, \tag{24}$$

where $0 < \alpha_{j,t} \leq 1$ may damp the penalty when the pattern retains strong positive evidence.

Revision eligibility. A scoped revision is proposed only after counterevidence is both sufficient and diverse. The generic predicate is

$$\mathrm{Revise}_t(p_j) = \mathbf{1}[C_{j,t} \geq \kappa_{\mathrm{ctr}}]\mathbf{1}[\rho_{j,t}^{\mathrm{ctr}} \geq \tau_{\mathrm{ctr}}]\mathbf{1}[N_{\mathrm{ind}}(\mathcal{D}_t(p_j)) \geq \kappa_{\mathrm{div}}]. \tag{25}$$

A revision created by Equation 8 begins as a candidate with fresh validation statistics. Its predecessor remains temporarily available for a bounded shadow period, preventing a local exception from immediately replacing previously useful guidance. Thus, downweighting precedes revision, and revision remains provisional during revalidation.

Candidate repair validation. The compatibility predicate χ in Equation 9 can combine semantic similarity with task- or environment-level grounding constraints. A newly propagated edge $e = (c', c_r) \in \mathcal{E}_t^{\mathrm{cand}}$ receives an initial score $v_0(e) = \varphi(s(c', c_f), \chi)$, where φ is bounded and monotone in similarity after compatibility is satisfied. Subsequent reuse maintains positive and negative evidence $H_t(e)$ and $D_t(e)$. A generic validation rule is

$$\mathrm{Valid}_t(e) = \mathbf{1}[H_t(e) \geq \kappa_{\mathrm{val}}]\mathbf{1}\left[\frac{D_t(e)}{\max(1, H_t(e) + D_t(e))} \leq \tau_{\mathrm{harm}}\right]. \tag{26}$$

Validated edges can later be revoked if sustained negative evidence exceeds a separate retirement criterion. Propagation therefore produces repair hypotheses, not observed failure–repair pairs or evidence that similarity alone guarantees transferability.

A.5 Adaptive Maintenance Scheduling

Pressure construction. For $o \in \{\mathrm{C}, \mathrm{R}, \mathrm{F}\}$, let $\mathbf{z}_{o,t}$ contain normalized changes in the signals relevant to operation o since its previous maintenance baseline. We compute

$$P_{o,t} = \mathrm{clip}_{[0,1]}\big(\boldsymbol{\beta}_o^{\top}\mathbf{z}_{o,t}\big)\,, \tag{27}$$

with configurable weights $\boldsymbol{\beta}_o$. For consolidation, $\mathbf{z}_{\mathrm{C},t}$ summarizes recurrence, positive evidence, candidate growth, and experience volume. For reconsolidation, $\mathbf{z}_{\mathrm{R},t}$ summarizes counterevidence, adverse reuse, and validation failure. For forgetting, $\mathbf{z}_{\mathrm{F},t}$ summarizes staleness, low applicability, interference, low utility, and capacity pressure. Count-based terms are normalized by an operation-specific evidence scale; all weights and normalizers are configuration parameters.

Stage transitions and periodic cadence. Let N_t^{cand} and N_t^{act} denote candidate and active pattern counts, and let V_t summarize accumulated reuse evidence. These statistics determine the memory stage σ_t (warmup, transition, or mature). Transition to a later stage requires configurable minimum evidence, while separate exit thresholds provide stage-level hysteresis. Stage-specific intervals support frequent organization during early accumulation and less frequent periodic maintenance as memory matures. The periodic component is

$$\mathcal{O}_t^{\mathrm{per}}(\sigma_t) = \begin{cases} \mathcal{O}_{\mathrm{enabled}}, & t - \ell_{\mathrm{per}} \geq T_{\sigma_t}, \\ \varnothing, & \text{otherwise}, \end{cases} \tag{28}$$

where $\mathcal{O}_{\mathrm{enabled}} \subseteq \{\mathrm{C}, \mathrm{R}, \mathrm{F}\}$ is the enabled operation set, ℓ_{per} is the previous periodic-maintenance step, and T_{σ_t} is the current stage's interval. Pressure-triggered operations complement this cadence when particular maintenance needs arise between periodic cycles.

Cooldown and hysteresis. The periodic and pressure-based sets in Equation 10 are combined subject to operation-specific execution controls. For last execution step ℓ_o and cooldown d_o, operation o is executable only if $t - \ell_o \geq d_o$, regardless of which scheduling route selected it. Pressure activations additionally use hysteresis: after activation, the pressure trigger is re-armed only when

Table 3: **Summary of experimental settings of benchmarks.**

Setting	Scope	#Tasks	Primary Metric
ALFWorld main	6 task families	6×500	Success Rate (Pass@1)
WebShop	Fashion stream	50	Final Score (after 2nd attempt)

$P_{o,t} < \eta_o - \delta_o$, with hysteresis margin $\delta_o > 0$. Periodic scheduling remains an independent source of maintenance candidates rather than requiring the pressure threshold to be crossed. A global minimum maintenance interval can additionally prevent consecutive sleep cycles. These controls regulate execution frequency without changing the operation-specific eligibility criteria for consolidation, reconsolidation, and forgetting.

B Experimental Details

This appendix specifies the benchmark construction, continual evaluation protocol, controlled variants, implementation settings, and measurement definitions supporting RQ1–RQ4.

B.1 Benchmarks and Continual Evaluation Protocol

Table 3 summarizes the experimental settings for ALFWorld and WebShop, including their evaluation scope, task scale, and primary metrics.

ALFWorld data. We use a balanced local ALFWorld export of 3,000 unique tasks, comprising 500 instances from each of six task families: `pick_and_place`, `clean_and_place`, `heat_and_place`, `cool_and_place`, `examine`, and `pick_two`. Each record provides the public task instruction, initial observation, task type, and a recorded action trajectory. To support long continual streams and sufficiently large memory accumulation, we use an open-loop protocol in which the model generates the complete action sequence in a single response, and a structural judge symbolically replays the sequence to determine whether the task is successfully completed. For evaluation, each task family is treated as an independent 500-task continual stream, with a separate run directory and memory initialized from scratch. Consequently, memory evolves only from the model's own online attempts and their public structural feedback within each family, rather than being shared across a mixed cross-family stream.

WebShop data and environment. The WebShop experiment replays a fixed 50-task fashion suite against the official WebShop simulator. Experiments use the same ordered task IDs as the pinned reference run, a 1,000-product catalog, and simulator goal seed. The runner stores both the ordered task IDs and a hash of the task manifest so that a resumed or repeated experiment cannot silently change the prompts. Each episode uses real page observations and official simulator reward; the simulator is reset between tasks, while the method's memory is retained across the 50-task stream.

Prequential continual protocol. All online methods follow a prequential protocol. Before task t, the method retrieves from the memory state produced by tasks $1, \ldots, t-1$ and generates a first attempt. The resulting trajectory and feedback are then recorded before the stream advances. A failed task is allowed **one additional attempt**, giving at most two attempts per task. Pass@1 always denotes the first attempt, while Pass@2 denotes the final outcome after this fixed retry budget.

B.2 Baselines and Controlled Variants

No Memory. No Memory uses the same task input, executor, judge, decoding configuration, and retry budget as the memory methods but disables all cross-task retrieval and memory updates. Its second attempt is therefore another model attempt with an empty memory context.

Case-only and Case-pattern. Case-only is the trajectory-grounded EXG-style control. It stores online cases, uses semantic case retrieval together with task anchors, `similar_to` expansion, and

verified `fixed_by` repair links, but does not maintain a separate pattern layer or run lifecycle management. Case-pattern keeps the same case machinery and adds online success/repair patterns and the basic consolidation path. It disables the subsequent long-term management package: reconsolidation, candidate-repair propagation, forgetting, freshness decay, pattern downweighting, case merging, and semantic retirement are all disabled. The Case-only → Case-pattern comparison therefore measures the joint contribution of abstraction and consolidation.

G-Dreamer. G-Dreamer retains the same executor, retrieval interface, and online pattern authoring as Case-pattern, then enables the complete lifecycle manager: adaptive maintenance scheduling, consolidation, reconsolidation, candidate-repair propagation and validation, freshness-aware downweighting, semantic compression/retirement, and forgetting. This makes the Case-pattern → G-Dreamer comparison a controlled test of the long-term management package while keeping the underlying task protocol fixed.

ExpeL. Our ExpeL adaptation preserves a two-phase protocol. For a stream of N tasks, the same task set is shuffled and the first 50% is used for collection. During collection, attempts and successful failure-to-success repair pairs are stored. At the phase boundary, the implementation performs offline LLM extraction over successful trajectories and verified repairs, semantically deduplicates the resulting rules at threshold 0.90, and retains at most five patterns. Memory is then frozen for the remaining 50% evaluation phase. The implementation also retrieves successful collection cases during evaluation. The reported run summary retains the full-stream Pass@1 average.

A-MEM and ReasoningBank. A-MEM stores one atomic note per attempt, generates note metadata, builds semantic links to nearby notes, and may evolve linked neighbors before later retrieval. ReasoningBank extracts up to three generalizable reasoning-memory items from each completed success or failure. In both adaptations, retrieval ranks ten semantic candidates and injects at most five memory items. Their states are persistent and updated online.

B.3 Evaluation Metrics

Task performance (RQ1). Let $y_{c,i}^{(1)}, y_{c,i}^{(2)} \in \{0,1\}$ denote success on the first attempt and after the fixed two-attempt budget, respectively, for the i-th task of ALFWorld family c. The family success rates are

$$\mathrm{SR}_c^{(a)} = \frac{1}{N_c} \sum_{i=1}^{N_c} y_{c,i}^{(a)}, \qquad a \in \{1,2\}, \tag{29}$$

and the equal-weight macro-average is

$$\mathrm{SR}^{(a)} = \frac{1}{6} \sum_{c=1}^{6} \mathrm{SR}_c^{(a)}. \tag{30}$$

Here $N_c = 500$ in the balanced six-family evaluation. We report $100\,\mathrm{SR}^{(1)}$ and $100\,\mathrm{SR}^{(2)}$ as Pass@1 and Pass@2. This equal-family macro average is also equal to the task-level average here because all six families have the same size.

For WebShop, we report Pass@1 and Pass@2 from the official full-success label, together with the mean official simulator reward. The logger stores reward on its native $[0,1]$ scale; when displayed as a 0–100 score, we multiply the mean reward by 100. Under the retry-enabled protocol, the reported final reward is the best official reward obtained within the two-attempt budget.

Memory utility and long-term degradation (RQ1). Let $P_{m,b}$ be the first-attempt success rate of method m in block b, expressed in percentage units, and $P_{0,b}$ that of No Memory on the same tasks. Figure 2 reports this diagnostic on an independent 500-task ALFWorld `heat_and_place` run with Qwen3-14B under a different random seed. Relative memory utility is

$$G_{m,b} = P_{m,b} - P_{0,b}. \tag{31}$$

For B blocks, peak-to-final degradation is

$$\mathrm{LD}(m) = \max_{1 \le b \le B} G_{m,b} - G_{m,B}. \tag{32}$$

Both $G_{m,b}$ and LD are measured in percentage points. LD measures loss relative to the method's own peak, not its absolute usefulness: a method with uniformly negative gains can still have zero LD. We therefore interpret LD together with final-block gain. This is a descriptive measure on the realized task stream rather than a causal estimate of forgetting on a fixed probe set.

Token cost and latency (RQ2). Let T_i and ℓ_i denote the logged total LLM tokens and task-loop wall time assigned to completed task i. We report

$$\mathrm{Tok} = \frac{1}{N}\sum_{i=1}^{N} T_i, \qquad \mathrm{Latency} = \frac{1}{N}\sum_{i=1}^{N} \ell_i. \tag{33}$$

The main table reports $\mathrm{Tok}/1000$, while the efficiency figure uses unscaled token counts and seconds. Logical LLM calls include task answering, retry answering, reflection/pattern extraction, and baseline memory-update calls when they occur; provider-internal HTTP retries are not counted as additional logical calls. ALFWorld task latency covers the task loop including answering, retry, memory updates, and lifecycle maintenance, while excluding dataset/model startup. WebShop uses the same completed-task denominator and includes answering, retrieval, online learning, and SleepOps, but excludes simulator/model startup and post-task checkpoint serialization. Retrieval time is additionally decomposed so graph selection/ranking can be reported with embedding time removed. Figure 4 reports token-cost and latency comparisons on ALFWorld Heat tasks with Qwen3-14B.

Memory size and the efficiency frontier (RQ2). For G-Dreamer variants, memory size is the number of latest non-forgotten graph nodes, i.e., case-like nodes (including retained semantic capsules when present) plus pattern nodes; graph edges are reported separately. For A-MEM and ReasoningBank, the corresponding count is the number of persistent baseline records, with nested ReasoningBank memory items additionally reported rather than silently treated as graph nodes. For ExpeL, the count contains collection records plus its separately selected offline patterns. These units are therefore structurally meaningful within a method but are not byte-equivalent across methods.

Paired transfer outcomes (RQ3). For variants A and B evaluated on the same N task outcomes, let $y_i^A, y_i^B \in \{0,1\}$ be the paired results. Table 2 reports these paired transfer outcomes on the 500-task ALFWorld `heat_and_place` stream using Qwen3-14B. Positive and negative transfer rates use all N tasks as their denominator:

$$\mathrm{PTR}(A\to B) = \frac{100}{N}\sum_{i=1}^{N} \mathbb{I}[y_i^A = 0 \wedge y_i^B = 1], \tag{34}$$

$$\mathrm{NTR}(A\to B) = \frac{100}{N}\sum_{i=1}^{N} \mathbb{I}[y_i^A = 1 \wedge y_i^B = 0]. \tag{35}$$

Thus, PTR is not the fraction of A's failures that are repaired, and NTR is not the fraction of A's successes that are lost. The table's FTG denotes their net paired gain:

$$\mathrm{FTG}(A\to B) = \mathrm{PTR}(A\to B) - \mathrm{NTR}(A\to B) = \frac{100}{N}\sum_{i=1}^{N}(y_i^B - y_i^A). \tag{36}$$

PTR and NTR are percentages of tasks, while their difference is a success-rate change in percentage points, not a relative percentage improvement. FTG equals the difference in task-averaged SR on the paired set; it need not equal a difference in the main table's six-family macro SR when the task set or family weights differ. These rates describe outcome changes between systems rather than the causal effect of an individual retrieved memory.

B.4 Implementation Details

Table 4 summarizes the main model and memory settings used in our experiments. The retrieval settings specify both the top-k candidate pools and the maximum numbers of case, pattern, and repair hints injected into the model context. The remaining limits control generation length and, when specified, the total size of the retrieved memory context.

Table 4: **Key implementation settings of model and memory.**

Setting	ALFWorld	WebShop
Temperature	0.2	0
Top-p	0.95	1.0
Maximum output tokens	800	800
Embedding model	`all-MiniLM-L6-v2`	
Case retrieval top-k	10	10
Maximum injected case hints	5	5
Pattern retrieval top-k	5	5
Maximum active pattern hints	2	5
Maximum candidate pattern hints	2	0
Maximum repair hints	3 (+1 candidate)	3 (+1 candidate)
Memory-hint character budget	3000	None
Pattern semantic deduplication	0.82	0.82

Online pattern construction. ALFWorld success patterns are deterministically abstracted from verified successful trajectories and require support from two distinct tasks before normal activation. A verified failure-to-success retry can additionally trigger post-task reflection and LLM extraction of one repair rule of at most 25 words. WebShop uses a five-item online pattern budget consisting of at most two repair, two progress, and one success pattern. A progress rule is extracted from an attempt with positive official reward and captures reusable action steps that make partial progress toward the task goal, without treating the trajectory as a full success. Success recipes can become retrievable after one verified task, whereas repair and progress rules require support from two independent tasks. In both benchmarks, the semantic pattern deduplication threshold is 0.82. Literal task-specific object IDs, product names, queries, or option values are removed from reusable pattern text.

Table 5: **Lifecycle operation settings.**

Setting	ALFWorld	WebShop
Nominal maintenance period	25	26
Consolidation pressure threshold	0.45	0.45
Reconsolidation pressure threshold	0.35	0.35
Forgetting pressure threshold	0.20	0.20
Consolidation cooldown	10	10
Reconsolidation cooldown	8	8
Forgetting cooldown	50	50
Pressure hysteresis	0.03	0.03
Pressure window	25	25
Warm-up maintenance period	15	15
Mature maintenance period	75	75
Active-pattern confidence threshold	0.62	0.50

Lifecycle configuration. The full system runs benchmark-specific lifecycle profiles while preserving the same management operators. Table 5 summarizes the main scheduling and control parameters, including maintenance periods, operation-specific pressure thresholds and cooldowns, phase-aware scheduling, and the confidence threshold for active patterns. These settings regulate when consolidation, reconsolidation, and forgetting are triggered and prevent frequent state changes during long-term memory evolution. WebShop additionally protects a verified success recipe from being demoted by a single unfinished or contradictory episode; demotion and reactivation require evidence from distinct tasks.

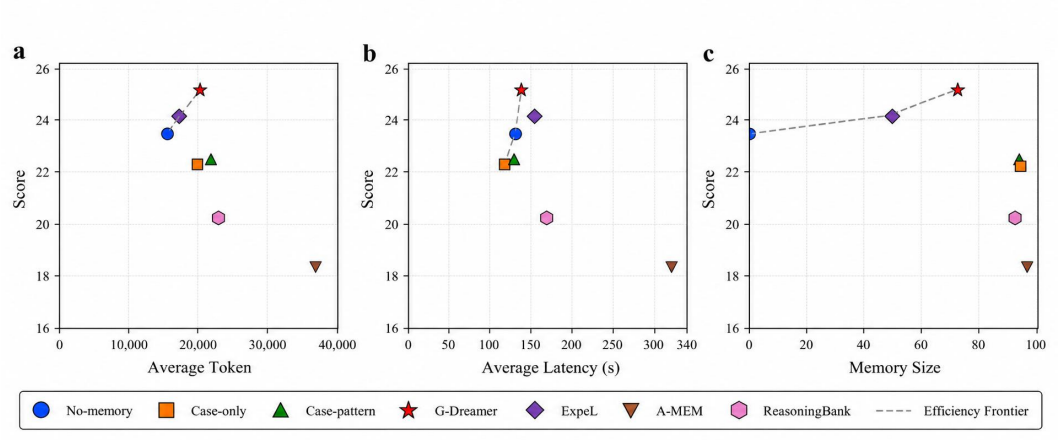

Figure 6: **Performance–efficiency trade-off on WebShop with Qwen3-8B.** Final score is compared with **(a)** average LLM tokens per task, **(b)** average task latency, and **(c)** final memory size.

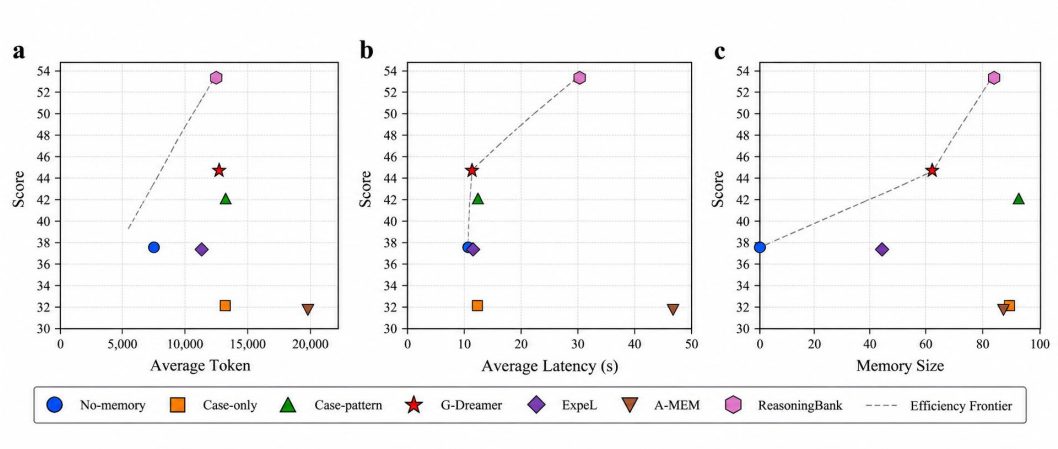

Figure 7: **Performance–efficiency trade-off on WebShop with Qwen3-14B.** We report final score against **(a)** average LLM tokens per task, **(b)** average task latency, and **(c)** final memory size.

C Additional Experimental Results

This section complements the main ALFWorld results with additional analyses on WebShop. We focus on two aspects that are not fully visible from the aggregate score alone: the trade-off between task performance and resource usage, and the evolution of memory utility along the continual task stream. For WebShop, score denotes the official simulator reward scaled to 0–100 after the fixed retry budget of at most two attempts.

C.1 WebShop Performance and Efficiency

Figures 6 and 7 compare final WebShop score against average token consumption, task latency, and final memory size. Token and latency statistics are averaged over all 50 tasks and include retries and memory maintenance, while memory size counts the final retained nodes or items.

Qwen3-8B. As shown in Figure 6, G-Dreamer achieves the highest score among the compared runs while remaining computationally competitive. Its token usage is comparable to the case-based variants, whereas its final memory state is substantially smaller than Case-only, Case-pattern, A-MEM, and ReasoningBank. The latency remains close to the simpler memory variants and is markedly below the heavier A-MEM and ReasoningBank baselines. This indicates that lifecycle

management improves the quality of the retained memory state without requiring memory growth proportional to the accumulated experience.

Qwen3-14B. Figure 7 further highlights the performance–efficiency trade-off with the larger backbone. Although ReasoningBank achieves the highest absolute score, G-Dreamer attains a competitive score with substantially lower latency and a smaller retained memory state. In particular, compared with ReasoningBank, G-Dreamer reduces average task latency by about 64% and final memory size by about 25%. It also improves over No-memory and Case-pattern while remaining close to the low-latency region of the efficiency frontier. These results show that G-Dreamer provides a favorable balance between task performance and the computational and memory costs required to sustain it.

C.2 Long-Term Score Dynamics on WebShop

We further examine whether the benefit of accumulated memory is preserved over the WebShop stream. Figure 8 divides the 50-task Qwen3-14B run into five consecutive 10-task blocks and reports the blockwise score gain over No-memory.

The accumulation-only variants exhibit substantially less stable utility. Case-pattern initially provides large gains ($+15.0$ and $+17.1$ points in the first two blocks), but falls to -16.8 in tasks 21–30 and finishes at only $+6.9$. Its peak-to-final degradation is therefore 10.2 points. Case-only also remains below No-memory over most of the stream and reaches only $+5.4$ in the final block. In contrast, G-Dreamer recovers from a smaller mid-stream drop (-4.6) and reaches its largest gain, $+16.0$, in the final block. Consequently, its peak-to-final degradation is zero while retaining a strong positive late-stream gain. Together with the ALFWorld diagnostic in the main paper, this suggests that lifecycle management is useful not only for controlling memory size, but also for preserving the utility of accumulated experience as the stream progresses.

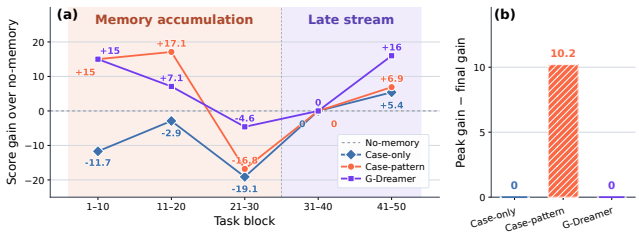

Figure 8: **Long-term memory utility on WebShop with Qwen3-14B. (a)** Blockwise final-score gain over No-memory. **(b)** Peak-to-final degradation. G-Dreamer preserves its strongest gain in the final block.