DeltaMem: Guarded Delta Memory for Safer Lifelong LLM Agents

Anonymous ACL submission

Abstract

LLM agents increasingly improve at test time by storing and reusing experience across episodes. Most lifelong-memory designs store successful trajectories or verbose reflections, which can be costly to inject and risky to reuse when the current state only partially matches a past episode. We introduce **DeltaMem**, instantiated here as **DeltaMem-lite**, a train-free memory mechanism that writes guarded *delta patches*: minimal corrective lessons distilled from failures with explicit applicability conditions. At decision time, reuse is treated as abstention, so the actor can apply a selected patch only when its guard fits or decline memory entirely. This design turns negative transfer from an aggregate performance drop into an auditable event: every selected memory item can be inspected through its trigger, guard, reuse reason, and counterfactual no-memory rerun. Controlled comparisons show that the value of lifelong memory depends jointly on the stored unit, reuse policy, and memory-token exposure. The resulting system provides a compact and interpretable operating point for safer reuse, while the symmetric Harm/Help/Net diagnostic reveals when retrieved experience helps, hurts, or should be disabled for a task distribution. DeltaMem therefore advances lifelong agent memory from storing more experience toward selective, guarded, and accountable reuse.

1 Introduction

Large language model (LLM) agents can improve at test time by accumulating experience across episodes without updating parameters, building on reasoning-and-acting paradigms such as ReAct (Yao et al., 2023c). Recent "lifelong" agent systems operationalize this by writing experience into external memory and reusing it later via retrieval and prompting (Ye et al., 2025; Yang et al., 2025; Liu et al., 2026; Yang et al., 2026; Zhang et al., 2026). However, the dominant memory units—successful trajectories and verbose reflections—are often long, instance-tied, and easy to overuse. In practice, they increase prompt cost and can induce *negative transfer* when reused out of context. Longer context windows do not solve this tension by themselves: interactive memory benchmarks and analyses show that models can underuse or misweight long histories, and that effective long-term memory requires *selective storage* and *selective reuse* (Liu et al., 2023; Wu et al., 2024; Maharana et al., 2024). For LLM agents, memory quality is therefore determined not only by what is stored, but also by how often that memory is exposed and whether the actor can decline to use it.

Goal and inspiration. Our goal is a train-free lifelong memory that *stores less* (lower prompt cost) and *reuses more safely* (reduces harmful carry-over) without amplifying negative transfer. Inspired by software debugging, we store not entire trajectories or long post-mortems, but the *minimal counterfactual change* that would have prevented a failure while leaving other decisions intact.

Delta memory. We distill each failure into a guarded *delta patch*: a trigger signature, the localized wrong pattern, a 1–3 sentence correction, and explicit preconditions/anti-conditions. At decision time, the agent retrieves candidate memory items and either applies one after checking applicability or abstains (`NONE`), casting reuse as abstention (Feng et al., 2024). In the conservative DeltaMem-lite system, this policy is top-1 with within-episode reuse suppression. In the matched-policy experiments, the same read-time policy is also applied to trajectory and reflection memories, allowing the effect of the stored unit to be analyzed with reuse exposure held fixed.

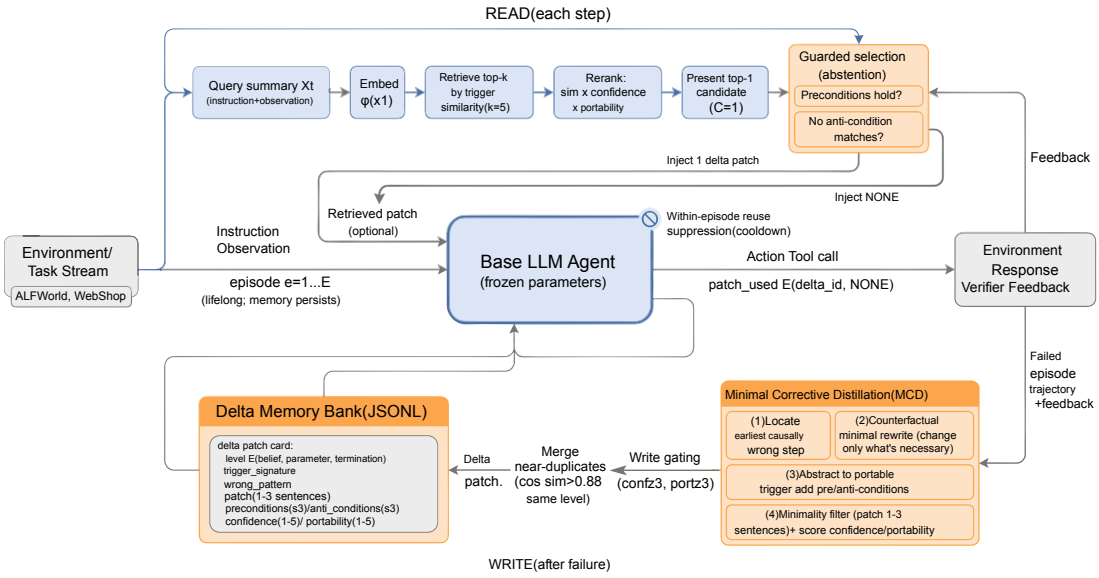

Figure 1: DeltaMem-lite overview. **Read:** at each step, retrieve a top-1 guarded delta patch and either apply it or abstain (NONE). **Write:** after failures, distill one new patch via minimal corrective distillation with write gating and merging.

Example. In WebShop-style tasks, a delta patch can simply say: "Before terminating, check for an explicit *finalize/submit* action; do not apply if the environment already signals completion and no further actions exist."

Contributions. We present a compact, fully train-free memory mechanism and an evaluation that separates storage, reuse, and exposure. The broader contribution is a shift from memory as accumulated prompt context to memory as a selective intervention whose use can be accepted, rejected, and audited:

- **DeltaMem-lite:** write *one* guarded minimal patch per failure; optionally deduplicate via trigger-based merging.
- **Minimal corrective distillation:** localize the first wrong step, apply a counterfactual minimal rewrite, and abstract into a portable patch with pre/anti-conditions.
- **Policy- and budget-controlled reuse:** compare trajectory, reflection, and delta memories under shared top-1 abstention and matched injected-memory budgets on WebShop, reducing read-time policy confounds while leaving write-time memory construction explicit.
- **Cost–harm evaluation:** controlled memory comparisons and conservative-system runs with success, steps, token cost, injected MemTok, memory use, selected-given-presented, and clearly scoped Harm/Help/Net rerun diagnostics.
- **Reuse diagnostics:** helpful/harmful case studies, failure-mode tags, patch-property correlations, confidence/portability validation, and a memory-off diagnostic for distributions where retrieved experience is net harmful.

2 Related Work

Lifelong LLM-agent memory. Experience-based improvement without parameter updates is increasingly studied for long-horizon agents, including hindsight reflection, reusable lessons, evolving skills, and lightweight memory systems (Ye et al., 2025; Yang et al., 2025, 2026; Zhang et al., 2026; Liu et al., 2026; Zhao et al., 2024). DeltaMem is complementary: it isolates the memory *unit* and asks whether minimal corrective patches can reduce exposure and make harmful reuse diagnosable.

Reflection, retrieval, and abstention. Reflection and self-correction improve language-agent behavior after failures (Shinn et al., 2023; Madaan et al., 2023; Yao et al., 2023b), while RAG and long-context work show that more retrieved text is

not always better (Lewis et al., 2020; Gao et al., 2023; Liu et al., 2023). Abstention and verification reduce unsupported outputs (Feng et al., 2024; Quan et al., 2024). DeltaMem transfers these ideas to agent memory: a retrieved item is an optional intervention with explicit guards, not mandatory background context.

Memory systems and evaluation. Memory is now treated as a first-class component of agent systems (Packer et al., 2023; Park et al., 2023; Liang et al., 2025), and interactive benchmarks highlight the need for selective storage and selective reuse (Wu et al., 2024; Maharana et al., 2024; Zhong et al., 2024; Chhikara et al., 2025). DeltaMem contributes a patch-centric design point: store less, expose less, and audit each selected reuse event.

3 DeltaMem

3.1 Problem setup

We consider a lifelong stream of episodes $e = 1, \ldots, E$ from an environment distribution. In episode e, the agent receives an instruction and interacts with the environment for up to T_e steps, producing actions a_t from the interaction history h_t. We study **train-free** improvement: the LLM parameters are fixed, and adaptation is achieved only through external memory read/write. The key design question is what memory unit should be stored, and how selectively that unit should be exposed at decision time.

3.2 Delta patch

A **delta patch** is a compact, reusable correction distilled from a failure. We focus on three local error types: `belief` (misinterpreting state), `parameter` (incorrect tool/slot/value), and `termination` (stopping too early or too late). Each patch stores: *trigger* (when to consider it), *wrong* (the erroneous pattern), *patch* (the minimal correction), *rationale*, and bounded *preconditions/anti-conditions* that prevent misuse. We also store two write-time quality signals: *confidence* and *portability*.

Levels. `belief` patches correct state interpretation errors (e.g., mistaking a disabled button for a transient failure); `parameter` patches correct local slot/value/tool-argument mismatches (e.g., format or granularity errors); `termination` patches correct premature stopping and missing finalization actions. These levels are intentionally local: they can be applied without rewriting the entire plan, and they admit explicit "do not apply when ..." guard conditions.

Schema constraints. To keep patches lightweight and reduce overfitting to instance details, we enforce a short patch length, bounded guard lists, and discrete confidence/portability scores. We represent each patch p as $p = (\ell, \tau, \omega, \delta, \mathrm{pre}, \mathrm{anti}, c, r)$, where ℓ is the level, τ is a trigger signature, ω is the localized wrong pattern, δ is the corrective text, and $\mathrm{pre}/\mathrm{anti}$ are bounded guards. The trigger τ is intended to capture the error mode, not merely topical similarity: task intent, compact observation summary, and the symptom that indicates the failure pattern.

3.3 Minimal corrective distillation

After a failed episode, DeltaMem writes at most one patch. Figure 2 summarizes the distillation procedure: (i) localize the earliest causally wrong step, (ii) propose a counterfactual *minimal* correction at that step, (iii) abstract away instance details into a portable trigger and add pre/anti-conditions, and (iv) enforce minimality and gate low-quality patches. This objective differs from generic reflection: it targets a *patch* that can be directly reused as a small decision-time constraint. Concretely, the distiller is instructed to keep all other decisions fixed and to change only what is needed to make the episode likely succeed, which prevents broad but irrelevant advice from being written into memory. The "first wrong step" objective is related to hindsight reflection (Ye et al., 2025) but differs in output granularity: we distill a guarded local correction rather than a reusable plan trace.

3.4 Memory bank, write gating, and merging

Patches are appended to a lifelong memory bank. To prevent redundancy, we embed patch triggers and **merge** a new patch into an existing patch of the same level when trigger similarity exceeds a threshold. We **gate** writes using minimum confidence and portability thresholds to avoid storing speculative or instance-tied patches. This selective write policy aligns with efficient lifelong memory designs that prioritize higher-information memory units over verbose histories (Liu et al., 2026).

Write score. Gating by confidence alone can still accumulate redundant or low-impact patches.

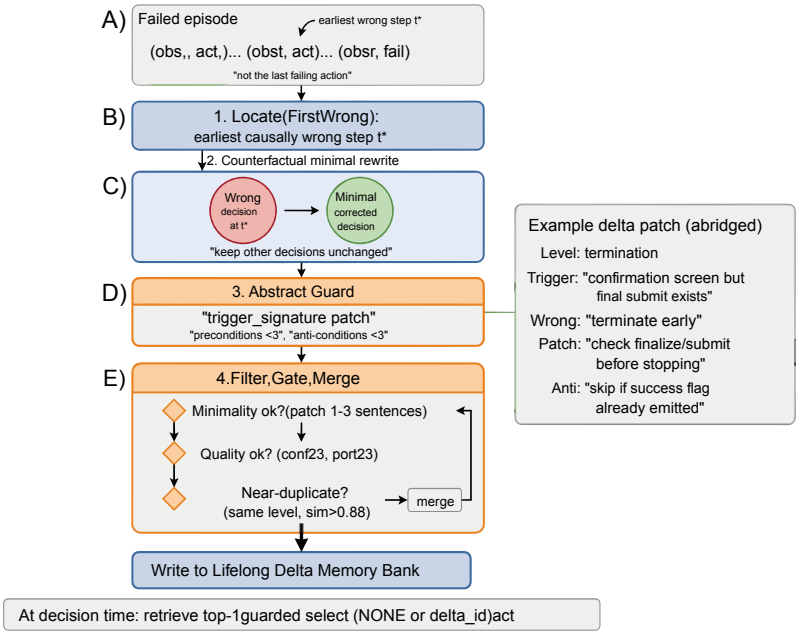

Figure 2: Minimal corrective distillation: localize the earliest wrong step, rewrite it with a minimal counterfactual correction, abstract it into a guarded trigger+patch, then filter, gate, and merge before writing to lifelong memory.

We therefore view writing as a utility–novelty trade-off and score each candidate patch p as:

$$s_{\text{write}}(p) = U(p) \cdot N(p) \cdot r_p \cdot c_p, \qquad (1)$$

where U estimates how much the patch would have helped, N is novelty with respect to existing patches, and r_p, c_p are portability and confidence. We write only when $s_{\text{write}}(p)$ exceeds a fixed threshold; otherwise the patch is discarded to control long-run memory quality. In practice, novelty is computed from trigger-embedding similarity, while portability and confidence are coarse LLM-judged quality signals.

3.5 Guarded reuse as abstention

At decision step t, the agent embeds a query summary of the current context and retrieves candidate patches by similarity to trigger embeddings. Candidates are reranked by a patch-aware score:

$$s(p \mid x_t) = \cos(\phi(x_t), \phi(\text{trigger}_p)) \cdot c_p \cdot r_p, \quad (2)$$

where $\phi(\cdot)$ is the embedding function. The actor may apply at most one selected patch, and reuse is optional: it must either justify that preconditions hold and anti-conditions do not match, or abstain with `NONE`. This casts memory use as an *abstention* decision rather than mandatory reuse (Feng et al., 2024). From a retrieval perspective, DeltaMem is a constrained form of RAG: it retrieves evidence-like items but applies them only under explicit guard conditions to reduce retrieval noise (Lewis et al., 2020; Gao et al., 2023).

Overuse control. Naively conditioning on multiple patches and repeatedly reusing the same patch can induce loops and harmful bias. DeltaMem-lite therefore uses sparse top-1 presentation and suppresses repeated use of the same patch within an episode. The matched-policy experiments apply the same read-time policy to trajectory and reflection memories, so the main comparison asks how the stored unit behaves once exposure is controlled.

Lifelong loop. DeltaMem alternates *act* and *write*:

1. **Act.** Embed the current context, retrieve and rank memory items, present candidates under the policy budget, then select one item or `NONE` before acting.

2. **Write.** After failure, distill one guarded delta patch, gate low-quality patches, and merge near-duplicates.

3.6 Why minimal and conservative patches help

Memory size and prompt cost. Assume DeltaMem writes at most one patch per failed episode and each patch has bounded length L. Then the memory bank grows as $O(F)$ items after F failures, and the memory prompt cost per decision is $O(CL)$ when conditioning on at most C candidates. By contrast, storing trajectory- or reflection-level memories with length proportional to episode horizon T yields memory growth and prompt overhead that scale with T.

Misuse exposure. In a simplified model where applying a retrieved item is harmful with probability at most q in the current context, conditioning on at most C candidates yields a worst-case harm probability bounded by Cq per decision. If patches are applied on at most m decision steps per episode due to abstention and reuse suppression, the episode-level misuse probability is at most mq. Thus, conservative conditioning and selective use directly reduce cumulative exposure to harmful memory.

4 Experiments

We evaluate on ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2023a) with sequential episodes and no memory reset. All variants use the same ReAct-style base actor (Yao et al., 2023c). We compare NoMem, TrajMem (successful-episode summaries), ReflMem (failure reflections), and DeltaMem (guarded delta patches).

Controlled suites. The matched-policy suite applies the same top-1 abstention and within-episode suppression policy to TrajMem, ReflMem, and DeltaMem, isolating the stored unit while holding read-time exposure fixed. The matched-budget suite additionally truncates injected memory to a common token budget. Conservative-system runs report DeltaMem-lite as a deployed operating point with sparse exposure; they are interpreted separately from unit-level comparisons. This separation is important because the original design goal has two parts: a compact memory unit and a conservative reuse policy. Without matched controls, an apparent gain could come from the policy rather than the delta patch, and an apparent failure could come from overexposure rather than the stored lesson itself.

Metrics and diagnostics. We report success, steps, total tokens, injected memory tokens (MemTok), memory use, and selection conditioned on presentation (Sel|Pres). For memory-used episodes, we rerun the same task with memory disabled and report Harm (memory run fails, no-memory rerun succeeds), Help (memory run succeeds, no-memory rerun fails), and Net (Help−Harm). This diagnostic makes negative transfer symmetric and attributable to selected reuse events. It is deliberately local: Success measures the whole run, while Harm/Help/Net asks what happened on the episodes where memory actually intervened.

5 Results

Table 1 reports the controlled WebShop comparisons. Under a shared top-1 abstention policy, TrajMem obtains the strongest raw success, while DeltaMem has the cleanest local Harm/Help/Net profile but lower end-to-end success. Under a matched memory-token budget, the ranking changes again: a shortened successful-trajectory memory is strongest on both success and local Net, while ReflMem and DeltaMem show harmful local rerun outcomes. This split is the central empirical finding. Raw success measures the whole policy-environment interaction, whereas Harm/Help/Net asks whether selected memory uses locally helped or hurt. Delta patches make selected reuse compact and inspectable, but gains depend on retrieval calibration and task fit. This makes the evaluation more diagnostic than a leaderboard comparison: when memory helps, the responsible item is visible; when it hurts, the selected trigger, guard, and reuse reason identify the failure point.

The conservative ALFWorld/WebShop runs give an end-to-end operating point rather than a unit-level causal claim. They show that sparse DeltaMem-lite reduces memory exposure relative to summary memories, especially on WebShop, but NoMem can remain the best raw-success system when the base actor is already strong. Thus memory should be treated as an optional mode: useful where sparse corrective reuse is needed, and disabled when retrieval exposure is not justified.

Suite	Method	Succ.↑	MemTok(k)↓	Use	Sel|Pres	Harm↓	Help/Net↑
matched-policy	NoMem	68.0	0.0	0.0	0.0	-	-
matched-policy	TrajMem	74.0	1.2	22.0	2.9	10.0	0.0 / -10.0
matched-policy	ReflMem	70.0	1.9	88.0	31.7	10.0	10.0 / 0.0
matched-policy	**DeltaMem**	48.0	2.0	72.0	18.3	0.0	10.0 / 10.0
matched-budget	NoMem	60.0	0.0	0.0	0.0	-	-
matched-budget	TrajMem	80.0	2.3	38.0	13.0	0.0	30.0 / 30.0
matched-budget	ReflMem	56.0	2.3	84.0	48.9	30.0	0.0 / -30.0
matched-budget	**DeltaMem**	52.0	2.0	58.0	26.4	30.0	0.0 / -30.0

Table 1: Core controlled WebShop results from accepted runs. The table evaluates memory behavior, not only aggregate winners: Use and Sel|Pres are exposure diagnostics rather than monotonic objectives, and Harm/Help/Net are symmetric no-memory rerun diagnostics over memory-used episodes. Full steps, token, and table-scope details are in Appendix A.6.

6 Analysis

The symmetric rerun diagnostic turns negative transfer into an observable reuse event. For each selected memory item, we can inspect the trigger, guard, reuse reason, and no-memory counterfactual outcome. Case studies show the intended pattern: helpful delta patches enforce local invariants, such as checking hard product constraints before changing variants; harmful patches often match the topic but not the decision state.

Patch-property diagnostics reinforce this interpretation. Reuse count and trigger specificity are most associated with Harm in the evaluated rerun cases, while LLM-judged confidence is nearly saturated and therefore weakly calibrated. This suggests that the primary risk is not only writing bad patches, but repeatedly applying plausible patches to nearby contexts. Accordingly, DeltaMem is best viewed as an execution-level safety layer: it is most useful when applications value low exposure, explicit guards, and post-hoc diagnosis, and it can complement broader strategy memories when tasks require high-level transfer. The key practical advantage is accountability: DeltaMem does not merely change the prompt, but records which past correction was offered, selected, and responsible for a downstream change in behavior.

The failure taxonomy and property analysis also clarify what still needs to improve. Repeated contextual mismatch points to retrieval and policy calibration, missing anti-conditions point to the written patch, and weak confidence calibration points to the need for verifier-backed scores. This decomposition is useful even when raw success is not highest: it turns an aggregate loss into concrete engineering targets for safer memory systems.

7 Discussion

When do minimal patches help? Delta patches are most effective when failures repeat with recognizable local signatures (e.g., recurrent state misinterpretations or recurring parameter formats). In these cases, a short "do *this* change" constraint can prevent repeated mistakes without importing irrelevant environment details. This setting is common in tool- and web-interaction tasks, where a small set of interface semantics (disabled buttons, slot formats, confirmation vs. completion) recurs across episodes. The useful delta patch behaves like a local invariant: it changes one decision while leaving the surrounding plan to the base actor. This explains why the helpful case in Table 10 is a verification rule rather than a new strategy.

When should memory abstain or switch off? When the base policy is already strong on a benchmark, retrieved memories can distract the agent or bias it toward suboptimal tool usage. The matched-budget WebShop run shows the other side of the same mechanism: a compact patch can still be harmful if its trigger matches the topic but the required action is different. This motivates two levels of abstention. Patch-level abstention decides whether a retrieved item applies at the current step; memory-off decides whether the task family should use external memory at all. The latter is necessary when a strong NoMem baseline and noisy retrieval make the expected value of memory negative.

What does summary memory capture? Successful-trajectory summaries can encode search priors that are useful in WebShop even after truncation: which product attributes to verify first, when to inspect variants, and when to continue search. Delta patches intentionally avoid such broad strategy transfer. This makes

them easier to audit, but it can underperform when the environment rewards coarse search heuristics more than local corrections. The results therefore separate two goals that are often merged: improving raw task success and making memory reuse small, guarded, and inspectable.

Design implication. The broader lesson is that lifelong memory should be evaluated as an intervention pipeline, not as a single prompt-augmentation trick. The stored unit determines what can be audited; the reuse policy determines how often memory can steer the actor; the budget determines how much retrieved text competes with the current observation. DeltaMem occupies the conservative end of this design space: it is most compelling when the application values low exposure, explicit guards, and post-hoc diagnosis of the selected memory item. For domains where broad strategy transfer dominates, delta patches are better viewed as an execution-level safety layer inside a larger memory system.

Relation to broader memory. DeltaMem-lite complements methods that learn reusable *skills* or evolving controllers across a broad task distribution (Yang et al., 2026; Zhang et al., 2026). It targets a narrower layer: a compact memory of guarded *corrections* that can be applied at test time with fixed parameters. For non-local failures where success requires a new subgoal decomposition or a multi-step strategy shift, richer hierarchical memories (e.g., planning vs. execution) may be necessary (Ye et al., 2025).

8 Conclusion

We introduced DeltaMem-lite, a train-free memory abstraction for safer lifelong LLM agents. Instead of storing more context, DeltaMem stores minimal corrective patches with explicit guards, applies them through abstention-style reuse, and evaluates them with symmetric Harm/Help/Net reruns. The central contribution is not another way to append experience to the prompt, but a disciplined memory unit that makes reuse an explicit, rejectable intervention. This combination makes memory reuse compact, inspectable, and diagnosable: when a past experience helps or hurts, the selected memory item and its applicability conditions can be audited directly. The results show that safe reuse is not a raw-success leaderboard alone, but an interaction among memory unit, reuse policy, memory-token exposure, and task distribution. DeltaMem therefore reframes lifelong agent memory around selective, auditable correction: useful agent memories should be compact enough to expose sparingly, guarded enough to reject, and explicit enough to explain both gains and failures. This design target is broadly valuable for future lifelong agents, and it leaves a clear path toward hybrids with broader strategy memories, stronger verifiers, cheaper distillers, and adaptive memory-off controllers.

Limitations

Our evaluation focuses on text-based agent benchmarks, and the controlled comparisons are designed as focused evidence for the memory-unit and reuse-policy questions rather than exhaustive deployment benchmarking. The run protocol is fixed by task order, logging schema, and table-specific inclusion criteria, so larger extensions can reuse the same evaluation pipeline directly. The baseline set uses controlled trajectory- and reflection-style memories to isolate the memory-unit question; richer lifelong-agent systems can be plugged into the same policy and diagnostic layer.

Delta distillation currently uses the same LLM family for acting and writing. Actor–distiller separation, verifier-backed guard checking, and calibrated confidence/portability scores are natural extensions because the patch schema already exposes the fields a verifier must inspect. Likewise, the failure taxonomy and memory-off controller are diagnostic layers rather than final production policies: manual annotation, wider rerun budgets, and prospective memory-off training would strengthen them without changing the central abstraction. Finally, DeltaMem-lite focuses on local corrective patches, not hierarchical goal or strategy memories; the two are complementary, and guarded delta patches can serve as the auditable execution-level component inside broader memory architectures (Quan et al., 2024; Feng et al., 2025; Peng et al., 2026).

Ethics Statement

This work studies general-purpose agent memory mechanisms in benchmark environments. We do not deploy agents in safety-critical settings. Our negative-transfer analysis highlights that experience reuse can harm performance; future deployments should include stronger validation and mon-

itoring before applying stored patches. In real-world use, external memories may inadvertently capture sensitive user content; practical systems should apply data minimization, redaction, retention limits, and access control. Finally, patch reuse can amplify undesirable biases or unsafe tool behaviors if triggers are underspecified; conservative selection should be paired with external safety policies and, where appropriate, human oversight.

References

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

Shangbin Feng, Weijia Shi, Yike Wang, Wenxuan Ding, Vidhisha Balachandran, and Yulia Tsvetkov. 2024. Don't hallucinate, abstain: Identifying llm knowledge gaps via multi-llm collaboration. *Preprint*, arXiv:2402.00367.

Tao Feng, Lizhen Qu, Niket Tandon, and Gholamreza Haffari. 2025. Iris: An iterative and integrated framework for verifiable causal discovery in the absence of tabular data. *Preprint*, arXiv:2510.09217.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*.

Jiafeng Liang, Hao Li, Chang Li, Jiaqi Zhou, Shixin Jiang, Zekun Wang, Changkai Ji, Zhihao Zhu, Runxuan Liu, Tao Ren, Jinlan Fu, See-Kiong Ng, Xia Liang, Ming Liu, and Bing Qin. 2025. Ai meets brain: Memory systems from cognitive neuroscience to autonomous agents. *Preprint*, arXiv:2512.23343.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. Simplemem: Efficient lifelong memory for llm agents. *Preprint*, arXiv:2601.02553.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *Preprint*, arXiv:2307.03172.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. *arXiv preprint arXiv:2402.17753*.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Preprint*, arXiv:2304.03442.

Bo Peng, Sirui Chen, Lei Xu, and Chaochao Lu. 2026. Causcientist: Teaching llms to respect data for causal discovery. *Preprint*, arXiv:2601.13614.

Xin Quan, Marco Valentino, Louise A. Dennis, and André Freitas. 2024. Verification and refinement of natural language explanations through llm-symbolic theorem proving. *Preprint*, arXiv:2405.01379.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. *Preprint*, arXiv:2010.03768.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *Preprint*, arXiv:2305.16291.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.

Cheng Yang, Xuemeng Yang, Licheng Wen, Daocheng Fu, Jianbiao Mei, Rong Wu, Pinlong Cai, Yufan Shen, Nianchen Deng, Botian Shi, Yu Qiao, and Haifeng Li. 2025. Learning on the job: An experience-driven self-evolving agent for long-horizon tasks. *Preprint*, arXiv:2510.08002.

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. 2026. Autoskill:

Experience-driven lifelong learning via skill self-evolution. *Preprint*, arXiv:2603.01145.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Preprint*, arXiv:2207.01206.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023b. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023c. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Shicheng Ye, Chao Yu, Kaiqiang Ke, Chengdong Xu, and Yinqi Wei. 2025. H^2r: Hierarchical hindsight reflection for multi-task llm agents. *Preprint*, arXiv:2509.12810.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026. Memskill: Learning and evolving memory skills for self-evolving agents. *Preprint*, arXiv:2602.02474.

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

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

A Experimental Details

A.1 Shared configuration

All methods use the same fixed base agent with no parameter updates. Unless otherwise specified, we use `gpt-5.4-mini` for acting and distillation, temperature 0. The preferred retrieval backend is `text-embedding-3-small`; when the embedding endpoint is unavailable, the code uses the recorded TF-IDF fallback for all methods in that run manifest. DeltaMem-lite uses top-k=5 recall, merge threshold 0.88, minimum confidence/portability 3 (1–5 scale), and a 1–3 sentence patch-length constraint. In the matched-policy main suite, TRAJMEM, REFLMEM, and DELTAMEM all use top-1 memory presentation, explicit abstention, and within-episode reuse suppression. In the matched-budget suite, all memory methods use the same retrieval/logging stack and a common injected-memory token budget. The conservative-system runs are reported only as operating-point evidence because they intentionally couple the delta memory unit with a stricter reuse policy.

A.2 Baseline details

NOMEM is the base ReAct-style agent without external memory (Yao et al., 2023c). TRAJMEM stores concise summaries of *successful* episodes, following successful-experience reuse in experiential agents (Zhao et al., 2024; Wang et al., 2023; Yang et al., 2026). REFLMEM stores concise structured reflections of *failed* episodes, following reflection-based improvement and hindsight reflection (Shinn et al., 2023; Ye et al., 2025). **DELTAMEM** stores one guarded minimal correction per failed episode and uses abstention-style top-1 application. To compare memory styles under a shared setup, all memory-based methods share the same acting backbone, retrieval stack, episode order, and prompting budget as closely as possible. We therefore treat TRAJMEM and REFLMEM as controlled internal baselines for memory *style*; they are not full reimplementations of ExpeL, H^{2}R, or SimpleMem under those projects' native protocols.

A.3 Backend and determinism

All reported runs use the OpenAI-compatible chat endpoint (`api_preference=chat`) with temperature 0. Our OpenAI-compatible endpoint exposes models as aliases such as `gpt-5.4-mini`, `gpt-5.2`, and `gpt-5.4`; it does not surface a provider-side snapshot identifier or a configurable reasoning-effort field in the saved run configs. As a result, we fix splits, prompts, run manifests, and outputs, but exact provider-side determinism still depends on the serving backend behind the alias.

A.4 Fixed splits, manifests, and run selection

Task orders are fixed by JSON split files prepared in advance (e.g., 50-episode and 100-episode splits), and all seeds for a given benchmark reuse the same task order. To avoid accidental drift from re-running methods multiple times, the paper tables and figures are generated from explicit run manifests, which enumerate the exact run directories used for aggregation. Quality checks require complete step/episode/memory-event logs, the memory-use schema, and rerun diagnostics matching each table's stated evidence scope. Single-benchmark tables are generated by the same manifest-selected pipeline, and their captions state the episode count, seed coverage, and benchmark scope. They are not pooled with runs from a different evidence scope.

A.5 Run selection and quality control

We separate execution from table eligibility. The main matched-policy and matched-budget tables are WebShop manifests (E=50, seed 0) that satisfy the quality checks. The conservative-system ALFWorld/WebShop tables are reported separately as operating-point evidence because they use a different reuse-policy regime. The quality checks require explicit manifest selection, complete episode/step/memory-event logs, the unified memory-use schema, evidence-scope-appropriate rerun diagnostics, DeltaMem case/property exports, and clean provider/runtime quality. These checks are deliberately conservative: runs with incomplete logs, duplicate cells, or provider/runtime errors are excluded from paper tables.

A.6 Full result tables

Tables 2 and 3 provide the full metric schema behind the compact main-text table. Tables 4 and 5 report the conservative-system operating point.

A.7 Token accounting

For acting, distillation, and embedding calls, we use API-reported token usage when available. For **MemTok**, we separately count only the retrieved-memory block injected into the decision prompt

Bench	Method	Succ.↑	Steps↓	Tok(k)↓	MemTok(k)↓	Use	Sell|Pres	Harm↓	Help↑	Net↑
WebShop	NoMem	68.0	10.8	12.4	0.0	0.0	0.0	-	-	-
WebShop	TrajMem	74.0	10.5	15.1	1.2	22.0	2.9	10.0	0.0	-10.0
WebShop	ReflMem	70.0	11.7	16.7	1.9	88.0	31.7	10.0	10.0	0.0
WebShop	**DeltaMem**	48.0	12.4	20.3	2.0	72.0	18.3	0.0	10.0	10.0

Table 2: Matched-policy comparison from the manifest-selected WebShop E=50, seed 0 runs. Single-seed values are shown without seed-level uncertainty. All memory methods use top-1 abstention with within-episode reuse suppression. Use and Sell|Pres are interpreted as exposure diagnostics rather than monotonic objectives. Harm/Help/Net are symmetric no-memory rerun diagnostics over memory-used episodes.

Bench	Method	Succ.↑	Steps↓	Tok(k)↓	MemTok(k)↓	Use	Sell|Pres	Harm↓	Help↑	Net↑
WebShop	NoMem	60.0	11.2	13.0	0.0	0.0	0.0	-	-	-
WebShop	TrajMem	80.0	9.5	14.8	2.3	38.0	13.0	0.0	30.0	30.0
WebShop	ReflMem	56.0	12.5	19.0	2.3	84.0	48.9	30.0	0.0	-30.0
WebShop	**DeltaMem**	52.0	11.0	18.7	2.0	58.0	26.4	30.0	0.0	-30.0

Table 3: Matched-budget comparison from the manifest-selected WebShop E=50, seed 0 runs. Single-seed values are shown without seed-level uncertainty. Summary memories and delta patches share a 300-token per-step injected-memory budget tier. Use and Sell|Pres are interpreted as exposure diagnostics rather than monotonic objectives. Harm/Help/Net are symmetric no-memory rerun diagnostics over memory-used episodes.

Method	Succ.	Steps	Tok(k)	Mem(k)	Use(%)	Neg(%)
NoMem	59.7 ± 3.3	29.0 ± 0.8	**22.6 ± 0.6**	**0.0 ± 0.0**	**0.0 ± 0.0**	-
TrajMem	62.0 ± 2.8	**26.7 ± 0.9**	30.5 ± 0.8	8.3 ± 0.2	86.7 ± 2.1	-
ReflMem	**63.3 ± 2.9**	27.2 ± 1.4	39.0 ± 2.0	16.1 ± 0.8	88.3 ± 2.9	-
DeltaMem	60.3 ± 2.5	28.5 ± 1.1	30.3 ± 1.2	6.2 ± 0.3	39.0 ± 7.5	13.3

Table 4: Multi-seed results on ALFWorld (mean ± std). Tok(k) and Mem(k) denote mean total tokens and retrieved-memory tokens per episode, in thousands. Use(%) is the percentage of episodes with selected memory. Neg(%) is the percentage of harmful reruns among evaluated reruns (available only for DeltaMem-lite in the scale setting; at most 10 reruns per seed).

Method	Succ.	Steps	Tok(k)	Mem(k)	Use(%)	Neg(%)
NoMem	**86.0 ± 1.4**	**7.5 ± 0.2**	**8.1 ± 0.2**	**0.0 ± 0.0**	**0.0 ± 0.0**	-
TrajMem	80.7 ± 3.9	8.2 ± 0.4	18.0 ± 0.8	3.9 ± 0.2	59.0 ± 4.5	-
ReflMem	69.0 ± 2.8	9.8 ± 0.1	22.4 ± 0.3	5.9 ± 0.2	94.3 ± 0.9	-
DeltaMem	76.3 ± 5.4	8.6 ± 0.6	17.2 ± 1.9	2.1 ± 0.3	27.3 ± 14.5	15.0

Table 5: Multi-seed results on WebShop (mean ± std). Tok(k) and Mem(k) denote mean total tokens and retrieved-memory tokens per episode, in thousands. Use(%) is the percentage of episodes with selected memory. Neg(%) is the percentage of harmful reruns among evaluated reruns (available only for DeltaMem-lite in the scale setting; at most 10 reruns per seed).

using `tiktoken`, so that total-token cost and memory-prompt cost can be analyzed independently.

A.8 Positioning against related memory methods

Table 6 summarizes the design axis emphasized by DeltaMem-lite relative to representative memory-based agents. DeltaMem-lite focuses on localized guarded corrections and explicit harmful-reuse diagnostics.

A.9 Embedding sensitivity (small A/B)

We performed a small sensitivity check comparing `text-embedding-3-small` vs. `text-embedding-3-large` on short runs (10 episodes, seed 0). Results can be noisy because early episodes may have few failures, yielding little memory to retrieve and making embedding choice hard to compare. In these short DeltaMem-only runs, `3-small` attains 70% success with 2.0k MemTok on ALFWorld, while `3-large` attains 60% success with 11.1k MemTok. On WebShop, `3-large` happens to incur no early failures and thus uses no memory (100% success, 0 MemTok), making the comparison inconclusive. For manifest-backed comparisons, we use one retrieval backend consistently across methods within each run manifest. When the embedding endpoint is unavailable, the code falls back to TF-IDF retrieval and records that backend in each run summary.

A.10 Symmetric no-memory rerun protocol

In the matched-control WebShop suites, for every memory method we select memory-used episodes and rerun the same task instance with memory disabled. We evaluate up to 10 such episodes per run by default (earliest occurrences), then report both directions of the counterfactual: Harm is an original failure followed by no-memory rerun success, while Help is an original success followed by no-memory rerun failure. Net is Help − Harm.

Method	Memory unit	Explicit guard	Abstention-style reuse	Harm diagnostic	Reference
Reflexion	verbal reflection after failure	No	No	No	Shinn et al. (2023)
ExpeL	reusable lessons / experience summaries	Limited	No	No	Zhao et al. (2024)
H^2R	hierarchical hindsight reflections	Limited	No	No	Ye et al. (2025)
SimpleMem	compressed lifelong memory items	Limited	Not central	No	Liu et al. (2026)
DeltaMem	guarded minimal corrective patch	Yes	Yes	Yes (subset reruns)	this work

Table 6: Positioning DeltaMem-lite against representative memory-based agents. DeltaMem-lite emphasizes a localized *memory unit*, guarded reuse, abstention, and subset-based harmful-reuse diagnostics.

Bench	Evaluated	Harm(%)	Help(%)	Net(%)
ALFWorld	30	13.3	6.7	-6.7
WebShop	28	14.3	0.0	-14.3

Table 7: Counterfactual help vs. harm on DeltaMem-lite rerun subsets in the conservative-system setting. Harm = original failure / no-patch rerun success. Help = original success / no-patch rerun failure. Net = Help - Harm.

This diagnostic is intentionally subset-based: it targets episodes where memory was actually used and therefore estimates local reuse effects rather than whole-benchmark causal effects.

A.11 Counterfactual help vs. harm (conservative-system reruns)

Table 7 reports the help/harm rates from conservative DeltaMem-lite rerun subsets. These numbers are useful for interpreting the operating point, while the main WebShop matched-policy/matched-budget tables report the same diagnostic symmetrically for TrajMem, ReflMem, and DeltaMem.

A.12 Auxiliary matched-policy check

As an additional policy-control check, we ran an E=50, seed 0 comparison in which TrajMem and ReflMem were forced to use the same top-1 reuse policy and within-episode suppression as strict DeltaMem (`inject_max_candidates=` 1, `patch_cooldown_steps=` 999). Table 8 gives a cross-benchmark auxiliary view with a smaller metric schema than the main WebShop matched-policy table. The ALFWorld row shows a small DeltaMem advantage under the shared top-1 policy, while the WebShop row shows no raw-success advantage over NoMem, TrajMem, or ReflMem. This auxiliary check is useful for interpreting benchmark sensitivity, while the main matched-policy table provides the complete Mem-Tok, Sel|Pres, and symmetric Harm/Help/Net schema.

Bench	Method	Succ.(%)	Tok(k)	Mem(k)	Use(%)
ALFWorld	NoMem	60.0	21.2	0.0	0.0
	DeltaMem	64.0	28.4	5.5	44.0
	TrajMem	62.0	25.1	2.0	56.0
	ReflMem	58.0	27.1	2.0	78.0
WebShop	NoMem	86.0	7.7	0.0	0.0
	DeltaMem	78.0	16.8	2.0	72.0
	TrajMem	86.0	11.8	1.2	16.0
	ReflMem	86.0	12.3	0.3	92.0

Table 8: Auxiliary matched-policy summary (E=50, seed 0). TrajMem and ReflMem are rerun under the same top-1 + reuse-suppression policy as strict DeltaMem.

Budg.	$s_{\min}$	Succ.	Tok(k)	Mem(k)	Use	Sel|Pres	Harm	Help	Eval
150	0.20	65.0	15.72	0.56	55.0	50.8	25.0	0.0	8
150	0.25	65.0	15.20	0.98	40.0	13.2	37.5	25.0	8
150	0.30	85.0	13.84	0.05	0.0	0.0	-	-	0
300	0.20	70.0	16.23	0.68	45.0	78.7	25.0	0.0	8
300	0.25	70.0	14.68	0.65	60.0	95.7	50.0	12.5	8
300	0.30	90.0	11.03	0.07	5.0	14.3	0.0	0.0	1

Table 9: DeltaMem-only exposure-control search on WebShop (E = 20, seed 0) under the matched-budget policy. $s_{\min}$ is the retrieval-score floor. Higher floors reduce memory exposure; this table is a calibration diagnostic, not the main multi-method comparison.

A.13 Exposure-control strategy search

We additionally ran a small DeltaMem-only WebShop search over the matched-budget token cap and a retrieval-score floor $s_{\min}$. This search is a calibration diagnostic rather than a method comparison: it asks how much memory exposure changes when the system refuses weak retrieval matches. Table 9 shows that lower floors produce substantially more memory use and can yield negative symmetric rerun outcomes, whereas $s_{\min}$ = 0.30 sharply reduces exposure and behaves close to a memory-off controller on this subset. Aggressive retrieval can be harmful even when the stored unit is compact, so retrieval floors and memory-off decisions are part of the memory design rather than implementation details.

A.14 Failure taxonomy and patch-property diagnostics

We export case-study bundles and patch-property rows for each evaluated DeltaMem run. Tables 10, 11, and 12 aggregate these rows across the WebShop matched-policy/matched-budget runs and the DeltaMem-only exposure search. The correlations are descriptive, not causal: they are computed only over memory-used cases selected for no-memory reruns. Table 13 separately bins the LLM-judged confidence and portability scores, showing why these scores should be treated as audit signals rather than calibrated safety certificates. The corresponding scripts also export per-case rows so that the automatic labels can be manually audited before making stronger qualitative claims.

A.15 Adaptive memory-off diagnostic

Table 14 in the main text reports a simple upper-level memory-off diagnostic on the WebShop matched-policy/matched-budget manifests. It illustrates the distinction between early calibration, full-run task success, and symmetric Net. The diagnostic is included to make the failure mode explicit: memory can look attractive under a narrow calibration slice while still requiring a memory-off decision once counterfactual harm/help is considered. We treat this as a retrospective diagnostic, not as a learned controller evaluated prospectively.

A.16 Distillation and conservative reuse constraints

DeltaMem-lite uses the LLM as both a distiller and a selector. We instruct the distiller to (i) localize the *first* causally wrong step, (ii) propose a counterfactual *minimal* correction at that step, and (iii) abstract it into a trigger with bounded pre/anti-conditions (dropping unrelated advice). At inference time, selection is abstention-style: the agent may choose `NONE`; otherwise it selects exactly one patch and justifies that preconditions hold and no anti-condition matches.

A.17 Prompt templates (abridged but executable)

The released code contains the exact prompt templates; here we summarize the fields that determine behavior.

Actor prompt. The actor is instructed to output JSON only with keys `action` and `patch_used`, to use at most one memory item, and to select `NONE` unless applicability is clear. The user prompt includes: environment hint, task instruction, current step index, current observation, the admissible action list (if any), and the retrieved memory block.

Delta distiller prompt. The distiller sees the truncated episode transcript and must return one JSON object with fields `level`, `trigger`, `wrong`, `patch`, `rationale`, `preconditions`, `anti_conditions`, `confidence`, and `portability`. Under the default minimal setting, the patch is capped at 1–3 sentences and guard lists are capped at 3 entries each.

Trajectory/reflection distillers. The successful-trajectory baseline writes at most 12 lines of imperative action summary. The reflection baseline writes at most 8 lines of failure reflection. These baselines are therefore concise, but they do not carry explicit anti-conditions or abstention-oriented guards.

A.18 Abstention dynamics (conservative-system scale runs)

We measure conservatism at the step level by the rate of selecting a memory item *conditioned on memory candidates being present*. Table 15 comes from the conservative-system scale runs. It describes the deployed conservative setting, where reuse policy and memory unit vary together across methods.

A.19 Conditioned memory cost (conservative-system scale runs)

Beyond selection, we measure the cost of surfacing memory by reporting (i) the mean number of candidates presented and (ii) mean MemTok injected per decision step, both conditioned on candidates being present. Table 16 again describes the conservative-system operating point. The matched-budget suite complements this view by fixing the injected-memory budget across memory styles.

In the same conservative-system scale runs (E=100), DeltaMem-lite stores 32.0 ± 3.3items on ALFWorld and 22.0 ± 5.7items on WebShop (mean$\pm$std over 3 seeds), reflecting one-patch-per-

Outcome	Source	Task context	Retrieved delta patch	Reuse decision	Rerun evidence
Help	matched-policy, ep. 5	WebShop task asks for men's drawstring gym shorts, army green, size 40, under a price cap.	**Trigger:** after opening a candidate, the agent may jump to variants or filters before verifying the selected item. **Patch:** first check title and option set against all hard constraints; only then change variants or continue search. **Anti:** do not apply if the page already clearly violates a hard constraint.	The actor selected the patch before clicking the product description, using it as a local verification step rather than as a new search strategy.	memory run succeeded; no-memory rerun failed
Harm	matched-budget, ep. 23	WebShop task asks for women's loose-fit tops in a specified color and medium size.	**Trigger:** search results look irrelevant or gender/category mismatched. **Patch:** refine the search before selecting a result. **Missing guard:** the patch did not distinguish useful refinement from over-constraining the query with noisy attribute text.	The actor rewrote the search query using many attributes at once; this steered the run away from a product that the no-memory rerun could recover.	memory run failed; no-memory rerun succeeded

Table 10: Concrete DeltaMem reuse cases selected from WebShop E=50, seed 0 case-study bundles. Help means the memory run succeeded while the no-memory rerun failed; Harm means the memory run failed while the no-memory rerun succeeded. The harmful case illustrates why anti-conditions must guard not only topical mismatch, but also over-specific query refinement.

Failure-mode tag	Count	Interpretation
trigger_mismatch	0	retrieved trigger only weakly matches the current context
missing_anti_condition	0	patch lacks a guard that should have blocked reuse
overgeneralized_patch	0	patch text is broad enough to steer unrelated tasks
repeated_contextual_mismatch	14	same item is reused in a context where it keeps steering the run
embedding_false_match	0	retrieval similarity surfaces a semantically wrong item
wrong_patch_level	0	belief/parameter/termination level does not match the current error
needs_manual_annotation	47	no automatic tag fired; inspect before treating as harmless

Table 11: Failure-mode taxonomy for evaluated DeltaMem reuse cases. Counts are automatically populated from logged fields and rerun outcomes; the manual-annotation row is conservative and should not be read as harmless.

Property	n	Corr(Harm)	Corr(Help)	HarmΔ	HelpΔ
trigger_specificity	61	0.33	0.13	0.04	0.03
reuse_count	61	0.34	-0.00	2.82	-0.06
portability	61	0.24	0.06	0.26	0.12
guard_density	61	0.20	-0.09	0.01	-0.01
retrieval_score	61	0.17	0.10	0.02	0.02
patch_tokens	61	-0.26	0.00	-4.68	0.08
precondition_count	61	0.12	-0.12	0.10	-0.18
anti_condition_count	61	0.08	-0.10	0.06	-0.14

Table 12: DeltaMem patch-property diagnostic over evaluated memory-used cases. Correlations are descriptive because confidence and portability are LLM-judged heuristics and reruns are subset-based.

Signal	Score	n	Harm	Help	Net
confidence	4	1	0.0	0.0	0.0
confidence	5	60	28.3	6.7	-21.7
portability	4	22	13.6	4.5	-9.1
portability	5	39	35.9	7.7	-28.2

Table 13: Validation diagnostic for LLM-judged confidence and portability over evaluated DeltaMem memory-used cases. High scores are not treated as calibrated safety certificates; Harm/Help/Net show how each score bucket behaves under no-memory reruns.

failure writing with write gating and optional deduplication.

A.20 Write gating and merging outcomes (conservative-system scale runs)

To make the write policy transparent, Table 17 reports the number of candidate patches distilled (approximately the number of failures), and how many are added, rejected, or merged. At E=100 in these conservative-system runs, merges are rare; bank size is therefore primarily controlled by conservative write gating rather than deduplication.

Suite	Bench	NoMem	BestMem	BestNet	Calib	Holdout	Guarded decision
matched_budget	WebShop	60.0	trajmem:80.0	trajmem:30.0	80.0	80.0	memory_on:trajmem
matched_policy	WebShop	68.0	trajmem:74.0	deltamem:10.0	80.0	40.0	memory_off:nomem

Table 14: Adaptive memory-off diagnostic from manifest-selected runs. BestMem reports the strongest memory method by full-run success, while BestNet reports the strongest symmetric rerun Net. The guarded decision enables memory only when the best-success memory method also passes the Net guard; this is a diagnostic upper-level policy, not a tuned production controller.

Method	ALFWorld	WebShop
TRAJMEM	30.1%	23.3%
REFLMEM	67.9%	89.3%
DELTAMEM	**2.9%**	**4.2%**

Table 15: Conservative-system step-level selection rate: fraction of decision steps where the agent selects a presented memory item (instead of NONE), conditioned on memory candidates being present (E=100, mean over 3 seeds).

Method	Cand	MemTok	Cand	MemTok
	ALFWorld		WebShop	
TRAJMEM	2.9	312	3.0	485
REFLMEM	2.9	612	2.9	623
DELTAMEM	**1.0**	**227**	**1.0**	**258**

Table 16: Conservative-system conditioned memory cost: mean number of candidates (**Cand**) and injected retrieved-memory tokens (**MemTok**) per decision step, conditioned on memory candidates being present (E=100, mean over 3 seeds).

Statistic	ALFWorld	WebShop
Candidates	39.7 ± 2.5	23.7 ± 5.4
Added	32.0 ± 3.3	22.0 ± 5.7
Rejected	7.3 ± 2.6	1.7 ± 0.9
Merged	0.3 ± 0.5	0.0 ± 0.0
Accept rate	81.4%	92.5%

Table 17: DeltaMem-lite write outcomes in the conservative-system scale runs (E=100, mean$\pm$std over 3 seeds).

B Additional Results and Analyses

B.1 Conservative-system learning curves and patch utilization

We provide additional curves, ablations, and qualitative examples to complement the main results. Figures 3 and 4 report multi-seed (mean±std, MA10) learning dynamics, retrieved-memory tokens, and patch utilization over episodes for the conservative-system setting.

B.2 Delta patch taxonomy and qualitative examples

Figure 5 reports the distribution of patch levels in the final DeltaMem bank in the E=50 runs. Belief patches dominate this bank, while parameter patches are comparatively sparse. Taken together, these local patch types suggest that many failures are driven by state interpretation and slot/value mismatches rather than global replanning. Table 18 provides representative delta items (abridged) highlighting minimal corrections with explicit anti-conditions.

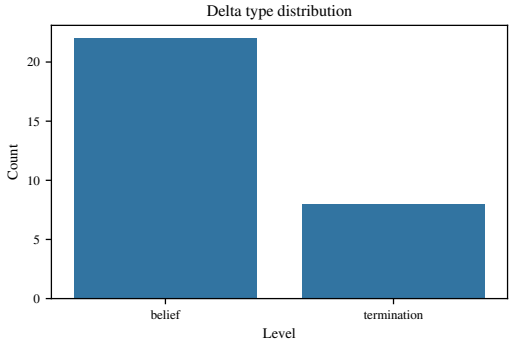

Figure 5: Distribution of delta levels in the final DeltaMem bank in the E=50 runs.

Type	Delta item (abridged)
Belief/ parameter	**Trigger:** interface expects a *city* slot but observation contains both an airport code and a city name. **Wrong:** fill destination with the airport code directly. **Patch:** normalize to a city-level destination; only use airport codes when explicitly requested. **Anti:** do not apply if the field explicitly says *airport* or the task requires terminal-level details.
Termination	**Trigger:** task appears complete but an explicit confirmation action remains. **Wrong:** stop on a confirmation screen without executing final submit. **Patch:** before terminating, check for explicit *finalize/submit*; stop only after an irreversible completion signal. **Anti:** do not apply if completion is already signaled (e.g., success flag) and no further actions exist.

Table 18: Representative delta patches with triggers, minimal corrections, and explicit anti-conditions (abridged).

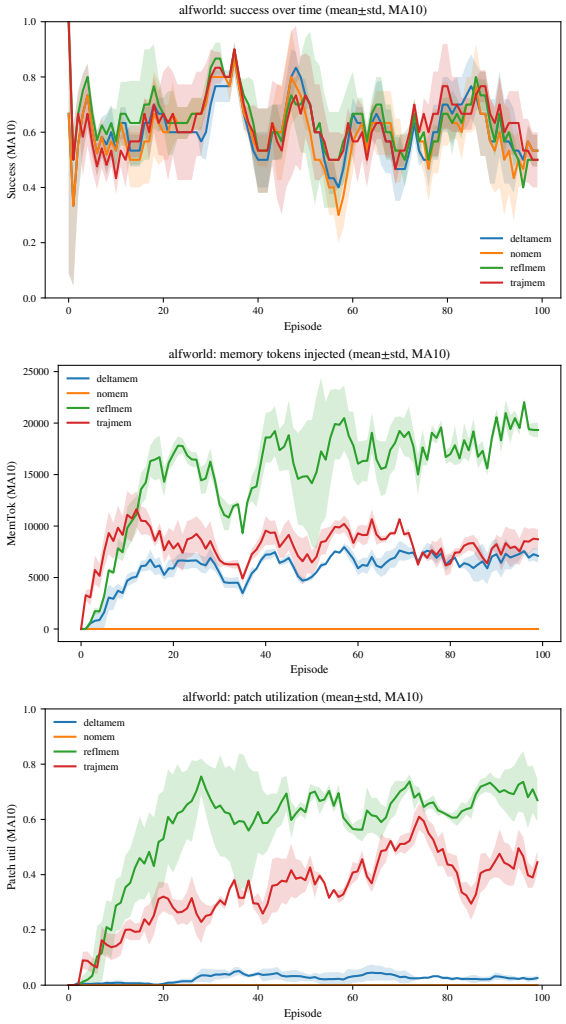

Figure 3: ALFWorld conservative-system scale curves: success (top), retrieved-memory tokens (middle), and patch utilization (bottom).

Qualitative pattern. Across both environments, the recurring reusable signal is usually not a full strategy but a narrow corrective rule: normalize a locally misread state, repair a slot/value choice, or delay termination until an explicit completion signal appears. Figure 5 and Table 18 are consistent with this view: `belief` patches dominate the evaluated bank, parameter patches are rare but conceptually important, and termination patches are fewer but often high leverage because they convert near-success traces into completed ones.

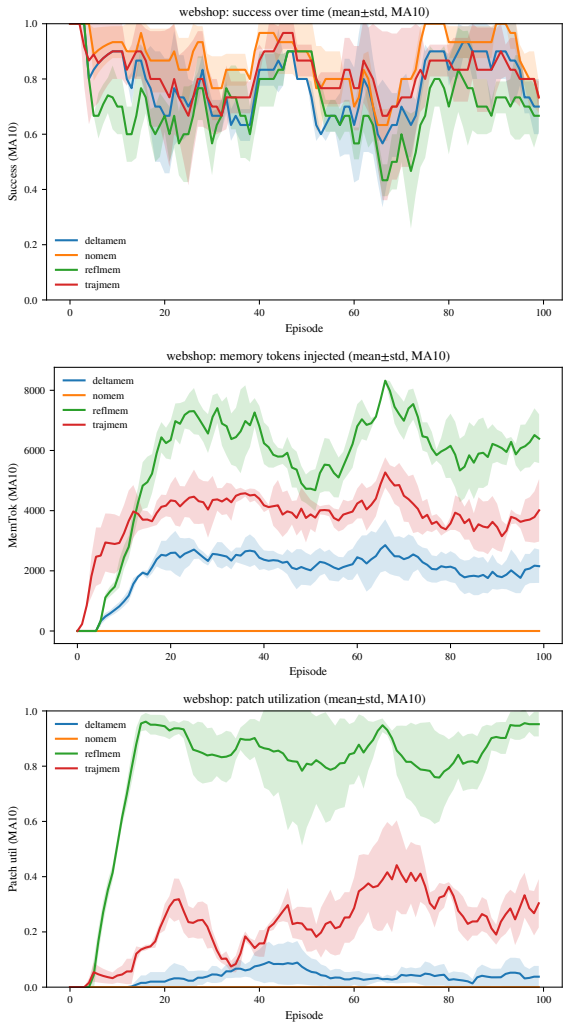

Figure 4: WebShop conservative-system scale curves: success (top), retrieved-memory tokens (middle), and patch utilization (bottom).

B.3 Design ablations (E=50)

Method	Succ.↑	Steps↓	Tok(k)↓	Mem(k)↓	Use↓	Neg↓
ALFWorld						
TrajMem	**66.0**	**25.5**	29.2	8.2	72.0	**0.0**
DeltaMem (-anti)	66.0	25.9	34.6	12.9	28.0	20.0
ReflMem	66.0	26.3	36.8	14.9	88.0	0.0
DeltaMem	64.0	27.4	40.1	17.2	80.0	0.0
DeltaMem (-minimal)	64.0	28.8	48.9	24.6	56.0	20.0
NoMem	60.0	27.4	**21.2**	**0.0**	**0.0**	-
WebShop						
ReflMem	**92.0**	8.4	15.1	3.0	74.0	**0.0**
TrajMem	90.0	**7.2**	15.8	3.5	50.0	10.0
NoMem	86.0	7.2	**7.7**	**0.0**	**0.0**	-
DeltaMem (-anti)	76.0	9.1	19.7	4.4	46.0	10.0
DeltaMem	70.0	9.8	23.0	6.3	44.0	40.0
DeltaMem (-minimal)	60.0	10.8	27.9	9.0	72.0	80.0

Table 19: Design ablations (E=50, seed 0).

B.4 Conservative application ablations

Table 20 compares looser DeltaMem variants against the stricter top-1 abstention-style configuration used in the main paper, which suppresses repeated within-episode reuse (cooldown).

Variant	Succ.↑	MemTok(k)↓	Use↓	NegTrans.↓
ALFWorld (baseline)				
DeltaMem	64.0	17.2	80.0	**0.0**
DeltaMem (-anti)	**66.0**	**12.9**	**28.0**	20.0
DeltaMem (-minimal)	64.0	24.6	56.0	20.0
ALFWorld (conservative)				
DeltaMem	64.0	5.5	44.0	0.0
DeltaMem (-anti)	**68.0**	**4.8**	70.0	0.0
DeltaMem (-minimal)	60.0	8.2	**42.0**	0.0
WebShop (baseline)				
DeltaMem	70.0	6.3	**44.0**	40.0
DeltaMem (-anti)	**76.0**	**4.4**	46.0	**10.0**
DeltaMem (-minimal)	60.0	9.0	72.0	80.0
WebShop (conservative)				
DeltaMem	78.0	2.0	72.0	10.0
DeltaMem (-anti)	82.0	**1.6**	**24.0**	10.0
DeltaMem (-minimal)	**86.0**	2.3	76.0	10.0

Table 20: Looser variants vs. the stricter main-paper application policy (top-1 retrieval with within-episode reuse suppression).

B.5 Distillation and selection prompts (abridged)

Distillation. For each failed episode, we instruct the LLM to: (i) localize the *first* causally wrong step, (ii) propose a counterfactual *minimal* correction at that step, and (iii) abstract it into a reusable delta item with bounded pre/anti-conditions. To avoid "reflection-style" verbosity, we explicitly penalize global replanning and require the patch to be 1–3 sentences.

Selection. At decision time, the agent is presented with at most one candidate patch (after retrieval and reranking) and must either apply it or abstain (`NONE`). We require a short justification that (i) preconditions hold and (ii) no anti-condition matches, making patch reuse auditable and discouraging opportunistic copying.