# Store Less, Reuse More Safely: Guarded Delta Patches for Train-Free Lifelong LLM Agents

**Anonymous ACL submission**

## Abstract

Test-time lifelong LLM agents often reuse past experience through full trajectories or verbose reflections. We study a more local alternative: many failures can be corrected by a small guarded change rather than by replaying an entire episode. We propose **DeltaMem-lite**, a fully train-free memory mechanism that writes one minimal corrective *delta patch* per failed episode and treats reuse as abstention. At decision time, the agent either applies a single retrieved patch after checking explicit guards or declines to reuse memory. Across ALFWorld and WebShop lifelong streams, DeltaMem-lite substantially reduces injected memory tokens relative to reflection-style memory while making harmful reuse directly observable through rerun-based diagnostics. Raw success gains remain benchmark-dependent, but among memory-based methods the approach yields a cleaner token–harm trade-off and supports more conservative, auditable 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).

**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 a top-1 candidate patch and either applies it after checking guards or abstains (`NONE`), casting reuse as abstention (Feng et al., 2024).

**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:

- **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.
- **Conservative patch application:** patch-aware ranking, `NONE`-allowed selection, and within-episode reuse suppression to reduce overuse.

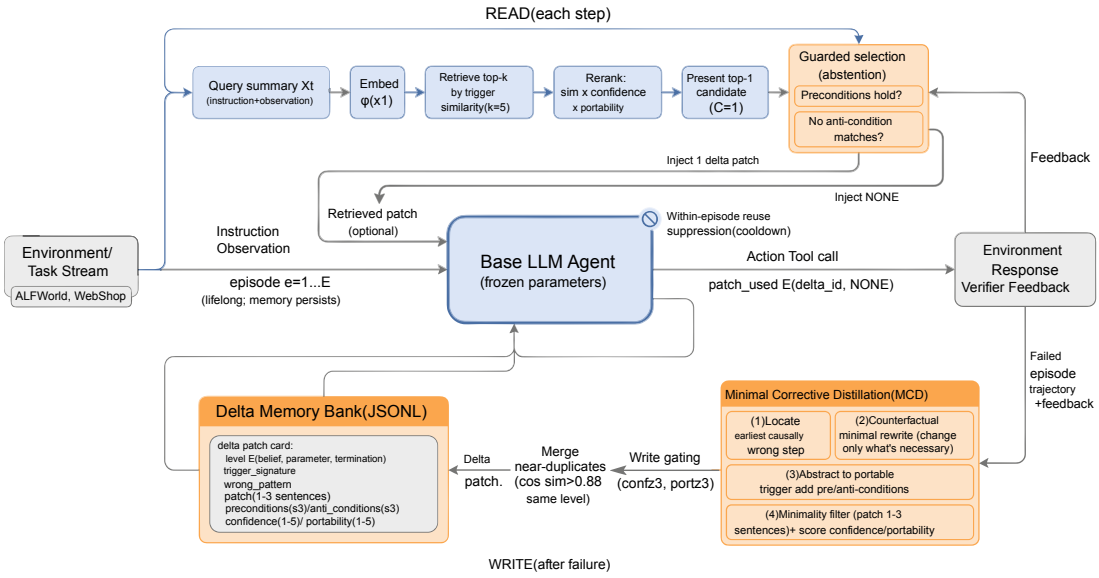


Figure 1: DeltaMem-lite overview. **Read:** at each step, retrieve a top-1 guarded delta patch and either apply it or abstain (NONE). **Write:** after failures, distill one new patch via minimal corrective distillation with write gating and merging.

- **Cost–harm evaluation:** ALFWorld and WebShop with success, steps, token cost, injected MemTok, patch use, and a rerun-based negative-transfer diagnostic.

## 2 Related Work

**Test-time lifelong improvement for agents.** Experience-based improvement without parameter updates is increasingly studied for long-horizon LLM agents. Hierarchical hindsight reflection ($H^2R$) organizes reflective memories across planning and execution levels (Ye et al., 2025). Other systems accumulate structured experience for long-horizon productivity tasks (Yang et al., 2025), or evolve higher-level skills and memory operations across episodes (Yang et al., 2026; Zhang et al., 2026). SimpleMem emphasizes efficient write/merge policies and retrieval for scalable lifelong memory (Liu et al., 2026). Related experiential learning agents store and retrieve past lessons to guide future decisions (Zhao et al., 2024). Our work is complementary: we focus on the *memory unit* itself and argue that a minimal *corrective* unit can improve the cost–harm trade-off of memory reuse, even when raw success gains are not universal.

**Reflection and self-correction.** Language agents can reflect on failures and use the reflection to improve subsequent behavior (Shinn et al., 2023), often within general reasoning-and-acting paradigms (Yao et al., 2023c). Related self-improvement paradigms include iterative self-refinement (Madaan et al., 2023) and structured deliberation (Yao et al., 2023b). DeltaMem-lite differs in that it distills a *localized* patch with explicit anti-conditions and enforces conservative application (select 0/1 patch).

**Conservative reuse, abstention, and verification.** Since retrieved memory can harm downstream decisions, memory use can be framed as an *abstention* problem: reuse should be avoided unless the retrieved item is clearly applicable (Feng et al., 2024). More generally, verification-based procedures refine model outputs using external checks (Quan et al., 2024), motivating our rerun-based negative-transfer diagnostic and our emphasis on guard conditions as first-class fields.

**Tool use and long-term memory.** Tool-augmented language models can learn to invoke external tools (Schick et al., 2023), and embodied agents can accumulate reusable behaviors over long horizons (Wang et al., 2023). Long-term memory mechanisms for LLM applications have

also been explored from a systems perspective (Packer et al., 2023) and in agent simulacra with memory-based behavior (Park et al., 2023). More generally, retrieval-augmented generation studies how retrieved evidence steers generation, with surveys emphasizing the relevance–noise trade-off in retrieval (Lewis et al., 2020; Gao et al., 2023). DeltaMem-lite targets a narrower question: what *minimal* experience unit is worth storing to prevent repeating specific mistakes.

**Abstraction and portability.** Polymorphic abstraction disentangles high-level intent from environment-specific implementations to improve generalization (Yu et al., 2026). Delta patches perform an analogous abstraction at the level of *error correction*: they aim to keep only what must change and strip instance-tied details.

**Memory as a systems perspective.** Surveys and taxonomies connect agent memory designs to cognitive inspirations and practical constraints (Liang et al., 2025). We contribute a concrete patch-centric design point that highlights a distinct trade-off axis: *store less* (minimal patches) to *reuse more safely* (avoid unnecessary carry-over while managing negative transfer).

**Long-context limits and memory evaluation.** Even with larger context windows, models can underutilize long histories and degrade on mid-context evidence (Liu et al., 2023). Interactive benchmarks such as LongMemEval and LoCoMo highlight that effective long-term memory requires both *selective storage* and *selective reuse* rather than simply appending more text (Wu et al., 2024; Maharana et al., 2024). Production-oriented systems (e.g., MemoryBank, Mem0) similarly emphasize scalable long-term memory with conservative retrieval (Zhong et al., 2024; Chhikara et al., 2025).

# 3 DeltaMem-lite

## 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 $a_t$ based on 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.

## 3.2 Delta patch

A **delta patch** is a compact, reusable correction distilled from a failure. We focus on three common, localizable error types: `belief` (misinterpreting state), `parameter` (incorrect tool/slot/value), and `termination` (stopping too early/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* (1–5).

**Levels (DeltaMem-lite).** `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: (i) a 1–3 sentence patch length constraint, (ii) at most 3 preconditions and 3 anti-conditions, and (iii) discrete confidence/portability scores in $\{1, \ldots, 5\}$.

**Notation.** We represent each patch $p$ as a tuple $p = (\ell, \tau, \omega, \delta, \text{pre}, \text{anti}, c, r)$, where $\ell$ is the level, $\tau$ is a trigger signature, $\omega$ is the localized wrong pattern, $\delta$ is the corrective patch text, and $\text{pre}/\text{anti}$ are bounded guard conditions. $c, r \in \{1, \ldots, 5\}$ denote confidence and portability.

**Trigger signature.** The trigger $\tau$ is a short signature intended for retrieval. We build it to capture not just topical similarity, but the *error mode*: task intent, a compact observation summary, and the specific symptom that indicates the failure pattern. This makes retrieval more targeted than matching on full trajectories, and enables merging and novelty estimates over triggers.

**Portability and confidence.** Portability $r$ rewards patches stated as semantic conditions (e.g., "field expects city-level value") rather than instance-tied strings (e.g., site- or UI-specific identifiers). Confidence $c$ reflects how well-localized and supported the diagnosis is (e.g., whether the

patch addresses the earliest wrong step and is consistent with episode feedback).

### 3.3 Minimal corrective distillation

After a failed episode, we distill exactly one patch. Figure 2 summarizes the distillation procedure: (i) localize the earliest causally wrong step (not the final mistake), (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 distillation 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 "helpful" 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 optional 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 $\tau_{\text{merge}}$ (Appendix). 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).

**Merging.** When two patches are near-duplicates, we retain the version with the higher portability and shorter corrective text (subject to the schema constraints), and aggregate provenance (source episodes) to support later auditing. In our current $E$=100 setting, however, merges are rare (Appendix), so write gating rather than merging is the primary driver of bank compactness.

**Write score (utility $\times$ novelty).** Gating by confidence alone can still accumulate redundant or low-impact patches. 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 (from failure diagnosis), $N$ is novelty w.r.t. existing patches (via trigger similarity), and $r_p, c_p$ are portability and confidence. We write only when $s_{\text{write}}(p) > \lambda$ for a fixed threshold $\lambda$, and otherwise discard the patch to control long-run memory quality. In practice, we implement novelty as $N(p) = 1 - \max_{p' \in \mathcal{M}} \cos(\phi(\tau_p), \phi(\tau_{p'}))$ over the current memory bank $\mathcal{M}$, and treat portability/confidence as coarse LLM-judged quality signals (Appendix).

### 3.5 Patch-aware retrieval and conservative application

At decision step $t$, we embed a query summary of the current context (instruction + observation) and retrieve top-$k$ candidate patches by cosine similarity to trigger embeddings. Candidates are reranked using 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, and $c_p, r_p$ denote confidence and portability. We then augment the agent context with at most $C$ candidates and require the agent to choose `NONE` or a single patch ID. To make pre/anti-conditions operational, selection must include a short justification that preconditions hold and no anti-condition matches; otherwise the agent must choose `NONE`. This casts patch use as an *abstention* decision rather than mandatory reuse (Feng et al., 2024). From a retrieval perspective, this is a constrained form of RAG: we retrieve evidence-like items but apply 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. We therefore adopt a conservative setting: condition on at most one candidate patch and suppress reuse of the same patch within an episode.

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

1. **Act (each step).** Embed the current context $x_t$ (instruction+observation), retrieve and rank patches, augment the context with top-$C$, then

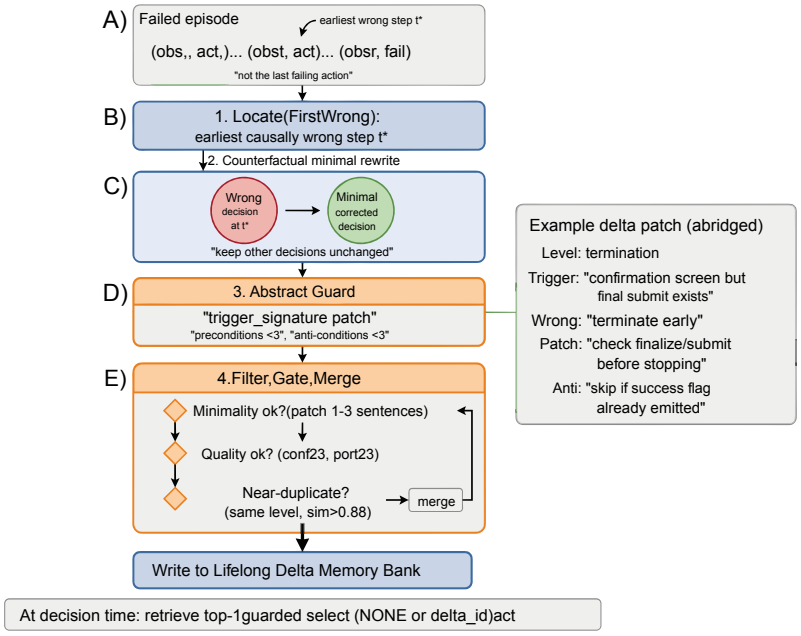


Figure 2: Minimal corrective distillation (MCD): localize the earliest wrong step, rewrite it with a minimal counterfactual correction, abstract into a guarded trigger+patch, then filter/gate/merge before writing to lifelong memory.

select `NONE` or one patch and act; suppress within-episode patch reuse.

2. **Write (after failure).** Distill one patch via minimal corrective distillation, gate low-quality patches, and merge near-duplicates.

### 3.6 Why minimal and conservative patches help

**Proposition (memory size and prompt cost).** Assume DeltaMem-lite writes at most one patch per failed episode and each patch has bounded length $L$ (in tokens). 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-level or reflection-level memories with length proportional to episode horizon $T$ yields memory growth and prompt overhead that scale with $T$.

**Proof sketch.** The item bound follows from the write policy (one patch per failure) and merging can only reduce the item count. The prompt-cost bound follows from the conservative conditioning cap and the bounded patch length.

**Proposition (overuse increases misuse risk).** 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 (union bound). If patches are applied on at most $m$ decision steps per episode (e.g., 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.

**Proof sketch.** Let $H_i$ be the event that candidate $i$ is harmful when applied. Then $\Pr[\cup_{i=1}^{C} H_i] \leq \sum_{i=1}^{C} \Pr[H_i] \leq Cq$. Applying the same argument over the $m$ steps where a patch is used yields $\Pr[\text{misuse in episode}] \leq mq$.

## 4 Experiments

### 4.1 Benchmarks and protocol

We evaluate on **ALFWorld** (Shridhar et al., 2021) (text household tasks) and **WebShop** (Yao et al., 2023a) (text shopping environment). Episodes are executed sequentially with **no memory reset** to simulate lifelong interaction. We report $E$=100 episodes per benchmark and aggregate over 3 random seeds. Model choices, embedding settings, retrieval thresholds, and the rerun protocol are deferred to the appendix.

### 4.2 Compared methods

We compare four train-free variants: NOMEM (the same ReAct-style base agent without external memory) (Yao et al., 2023c), TRAJMEM (store short summaries of *successful* episodes, following successful-experience reuse) (Zhao et al., 2024; Wang et al., 2023), REFLMEM (store short structured reflections of *failed* episodes, following reflection-based memory) (Shinn et al., 2023; Ye et al., 2025), and **DeltaMem-lite** (store one guarded delta patch per failure). All methods share the same base agent and differ only in what is written and retrieved. To compare memory styles under a shared backbone, all memory-based methods use the same retrieval interface as closely as possible; appendix gives the exact prompts, budgets, and conservative-application variants. Accordingly, TRAJMEM and REFLMEM should be read as *controlled surrogates* for successful-experience and reflection-style memory, rather than as full reproductions of larger systems such as ExpeL, $H^2$R, or SimpleMem.

### 4.3 Metrics

We report: (i) **Success** rate, (ii) average **Steps**, (iii) total API **Tokens** per episode (acting + write-time distillation), (iv) injected **MemTok** (prompt tokens attributable to retrieved memory), and (v) **MemUse** (fraction of episodes where retrieved memory is used). We include both total token cost and memory-only cost to separate prompt efficiency from downstream task outcome.

**Negative transfer (subset rerun).** To estimate harmful reuse, we additionally report a rerun-based negative-transfer diagnostic on a small subset. The full protocol is deferred to the appendix; in short, we revisit early patch-used episodes with the memory snapshot fixed and test whether disabling patches reverses failures.

## 5 Results

Tables 1 and 2 summarize lifelong results at $E$=100 episodes (mean±std across seeds; *Neg* is a rerun-based diagnostic and may be unavailable for some methods).

**Store less: memory prompt cost.** Across both environments, DeltaMem-lite cuts injected memory tokens (MemTok) by 2.6× on ALFWorld (16.1k→6.2k) and 2.8× on WebShop (5.9k→2.1k) relative to reflection memory. This efficiency comes from a bounded patch schema and conservative application: at most one candidate patch is considered, and abstention (NONE) is allowed. Consistent with this design, DeltaMem-lite uses memory in only 27–39% of episodes, whereas reflection/trajectory memories are used in most episodes (59–94%). At the decision-step level, DeltaMem-lite selects a patch in only 2.9%(ALFWorld) and 4.2%(WebShop) of steps when memory candidates are present (Appendix Table 6). Conditioned on candidates being present, DeltaMem-lite injects only 227(ALFWorld) and 258(WebShop) MemTok per decision, compared to 612/623for reflection memory (Appendix Table 7). In these $E$=100 runs, the resulting DeltaMem banks remain compact (32.0±3.3items on ALFWorld; 22.0 ± 5.7items on WebShop), showing that one-patch-per-failure writing plus gating is sufficient to keep memory small at the current scale.

| 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 1: 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 2: 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).

**Reuse more safely: trade-off rather than universal gains.** On ALFWorld, DeltaMem-lite roughly matches NOMEM in success (60.3 vs. 59.7) while using fewer total tokens than reflection memory (30.3k vs. 39.0k), but it does not outperform the stronger trajectory/reflection baselines on raw success. On WebShop, NOMEM is the strongest baseline; all memory variants re-

duce success, suggesting that retrieved experience can act as a distractor when relevance is imperfect. Within memory-based methods, DeltaMem-lite is less harmful than reflection memory (76.3 vs. 69.0) while incurring 2.8× fewer MemTok. DeltaMem-lite also makes misuse measurable: under a patch-triggered rerun protocol, we estimate 13–15% negative-transfer events on early patch-used episodes, which makes guarded patches (anti-conditions) and abstention-style selection necessary. We therefore read the main result as a better *cost–harm trade-off* among memory-based methods, rather than as evidence of universal success gains over no-memory agents.

# 6 Analysis

## 6.1 Learning dynamics and memory cost

Appendix Figures 3 and 4 show scale learning curves, retrieved-memory tokens (MemTok), and patch utilization over time. Trajectory and reflection memories tend to be used in most episodes, increasing MemTok. DeltaMem-lite is more selective, conditioning on fewer candidates and using them sparsely (Appendix Tables 6–7). Together with the low patch-use rates in Tables 1–2, this suggests that in these benchmarks "useful" memory is concentrated in a small set of recurring local mistakes, and that unconditional reuse is an inefficient default.

## 6.2 Ablation: conservative application

Appendix Table 11 compares looser pilot DeltaMem variants against the stricter top-1 abstention-style reuse policy adopted in the main paper. The stricter setting substantially reduces memory prompt cost and can improve success on WebShop, consistent with the hypothesis that patch *overuse* is a major failure mode. In pilot ablations, moving to the stricter configuration cuts MemTok by about 3× (e.g., 17.2k→5.5k on ALFWorld; 6.3k→2.0k on WebShop) and can improve WebShop success (70→78), supporting conservative application as a practical default when negative transfer is costly.

## 6.3 Negative transfer: when can patches hurt?

Delta patches are local, but they can still cause harm when their triggers are under-specified or when a patch is repeatedly applied in contexts that partially match but violate an anti-condition. This is most visible on WebShop, where NoMem is already strong and any distraction can reduce success. Conservative application mitigates these failures by limiting the patch to a single, high-confidence candidate and by preventing repeated reuse loops within an episode. These observations motivate two design choices: (i) storing explicit anti-conditions as first-class fields, and (ii) treating patch selection as an abstention problem (choose `NONE` unless conditions are clearly satisfied). This matches findings from retrieval-augmented generation: retrieved content can introduce noise and bias when relevance is imperfect (Lewis et al., 2020; Gao et al., 2023). Appendix Table 4 makes this counterfactual more symmetric: on the currently evaluated rerun subsets, help events do exist on ALFWorld, but harmful carry-over still exceeds helpful carry-over overall, and the net effect is negative on WebShop.

## 6.4 Delta patch taxonomy (pilot)

Appendix Figure 5 shows the distribution of patch types in the final memory bank on pilot runs. Belief patches dominate, while parameter patches are comparatively rare in the current pilot bank; together they indicate that many failures are caused by local state interpretation and slot/value formatting errors rather than global replanning.

## 6.5 Qualitative examples

Appendix Table 9 shows representative delta patches distilled from failures. The key property is *locality*: patches specify a minimal change and include anti-conditions that prevent reuse in mismatched contexts.

## 6.6 Pilot results and minimality/anti-condition ablations

To isolate design choices, we report pilot results (E=50, seed=0) and two ablations: removing anti-conditions (`-anti`) and relaxing minimality constraints (`-minimal`) in Appendix Table 10. We find that relaxing minimality can substantially increase MemTok and negative transfer (e.g., WebShop `-minimal` reaches 9.0k MemTok and 80% negative transfer under our subset metric), showing that "store less" is also a safety-relevant regularizer for lifelong reuse, not just a cost objective.

# 7 Discussion

**When do minimal patches help?** Delta patches are most effective when failures repeat with recog-

nizable local signatures (e.g., recurrent state misinterpretations or recurring parameter formats). In such tool- and web-interaction settings, a short "do *this* change" constraint can prevent repeated mistakes without importing irrelevant environment details, making memory reuse cheaper and easier to control than verbose reflection memories.

**Why can memory hurt?** When the base policy is already strong on a benchmark, retrieved memories can distract the agent or bias it toward suboptimal tool usage. This motivates patch-aware retrieval, explicit anti-conditions, and conservative application. Our results on WebShop emphasize that the value of lifelong memory depends on both the task distribution and the cost of misgeneralization; in such settings, NoMem can remain the best operating point.

**What delta patches do *not* replace.** DeltaMem-lite is not a substitute for learning reusable *skills* or evolving controllers across a broad task distribution (Yang et al., 2026; Zhang et al., 2026). Rather, it targets a narrower layer: a compact memory of guarded *corrections* applied at test time with fixed parameters. For non-local failures that require new subgoal decompositions or multi-step strategy shifts, richer hierarchical memories may be necessary (Ye et al., 2025).

## 8 Conclusion

We introduced DeltaMem-lite, a train-free lifelong memory mechanism that stores *minimal corrective patches* rather than full trajectories or verbose reflections. Delta patches distill failures into compact, reusable constraints with explicit anti-conditions, and are applied conservatively to control negative transfer. Across ALFWorld and WebShop, patch-centric memory substantially reduces injected MemTok (up to $2.8\times$ relative to reflection memory) and enables direct rerun-based diagnosis of harmful reuse. Our evidence suggests that DeltaMem-lite offers a better *cost–harm trade-off* for memory-based agents, rather than universal raw-success gains. The matched-policy pilot further suggests that unit-level effects are benchmark-dependent: DeltaMem retains a small success edge on ALFWorld, but the earlier WebShop advantage disappears once reuse policy is matched. An important next step is to isolate memory-unit effects from reuse-policy effects under matched budgets, and to extend guarded delta memories to hierarchical goal/plan and execution patches with stronger verifiers and cheaper distillers.

## 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 monitoring 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.

## 9 Limitations

Our evaluation is limited to two text-based benchmarks and a small number of seeds. Our main end-to-end comparison also couples the memory *unit* with a more conservative reuse policy for DeltaMem-lite (top-1, abstention, reuse suppression), so the current experiments do not fully isolate unit granularity from application policy under matched budgets. Stronger factorized comparisons—e.g., short/guarded reflection memories under the same top-1 abstention policy—are needed before making stronger causal claims about the delta unit alone. Indeed, the matched-policy pilot in Appendix Table 5 already shows benchmark-dependent behavior: DeltaMem retains a small ALFWorld success edge, but loses the earlier WebShop advantage once reuse policy is matched. Delta distillation is performed by the same LLM used for acting; separating distillers, validators, and actors (and mixing model sizes) is an important direction. Moreover, the distiller relies on natural-language rationales, whose faithfulness is imperfect; stronger validation and unfaithfulness-aware evaluation would improve reliability (Tutek et al., 2025). Our negative-transfer metric is a subset rerun test and likely underestimates long-tail harms; scaling this evaluation is future work. Finally, DeltaMem-lite currently focuses on local corrective patches; extending to higher-level goal/plan patches and adding verifier-backed refinement could improve transfer on com-

plex web settings (Quan et al., 2024; Feng et al., 2025; Peng et al., 2026).

## 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.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Preprint*, arXiv:2302.04761.

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.

Martin Tutek, Fateme Hashemi Chaleshtori, Ana Marasović, and Yonatan Belinkov. 2025. Measuring chain of thought faithfulness by unlearning reasoning steps. *Preprint*, arXiv:2502.14829.

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. $H^2$r: Hierarchical hindsight reflection for multi-task llm agents. *Preprint*, arXiv:2509.12810.

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. 2026. Polyskill: Learning generalizable skills through polymorphic abstraction. *Preprint*, arXiv:2510.15863.

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-5.1` for acting and distillation, temperature 0, and `text-embedding-3-large` for retrieval in the main runs. 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 main scale runs, DeltaMem-lite uses top-1 memory injection with within-episode reuse suppression, while the trajectory/reflection baselines use top-3 injection in the main comparison. This asymmetry is intentional for the current end-to-end system, but it also means that the main table does not fully isolate memory *unit* effects from reuse-policy effects; we call this out explicitly in the limitations.

## 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. The current gateway exposes the model as the alias `gpt-5.1`; it does not surface a provider-side snapshot identifier or a configurable reasoning-effort field in the saved run configs. As a result, we can lock the *artifact* (splits, prompts, manifests, run 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 pilot 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 frozen manifests included with the artifact, which enumerate the exact run directories used for aggregation. For the scale tables, these manifests resolve to 3 seeds per benchmark and method.

## A.5 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 using `tiktoken`, so that total-token cost and memory-prompt cost can be analyzed independently.

### A.6 Positioning against related memory methods

Table 3 summarizes the design axis emphasized by DeltaMem-lite relative to representative memory-based agents. Our intended contribution is not a universal stronger agent, but a more localized and guarded memory unit with an explicit harmful-reuse diagnostic.

### A.7 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 the main experiments, we use `text-embedding-3-large` consistently across methods.

### A.8 Negative transfer rerun protocol (subset)

For episodes where a delta patch is used, we rerun the same task instance with patches disabled while holding the memory snapshot fixed. We evaluate up to 10 such episodes per run (earliest occurrences) and count a rerun-success / original-failure as a harmful reuse event. This diagnostic targets early failures where patches are most likely to be reused repeatedly, and it underestimates long-tail harms.

### A.9 Counterfactual help vs. harm (scale DeltaMem runs)

The rerun protocol also exposes the opposite event: if the original DeltaMem run succeeds but the no-patch rerun fails, then memory was locally *helpful* on that evaluated episode. Table 4 reports both rates on the currently available scale reruns.

### A.10 Matched-policy pilot

To partially factorize memory *unit* from reuse policy, we ran an additional pilot ($E$=50, seed 0) 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 5 shows a benchmark-dependent pattern. On ALFWorld, strict DeltaMem retains a small success advantage over matched TRAJMEM (64.0 vs. 62.0), but not a token advantage (28.4k vs. 25.1k total tokens). On WebShop, the earlier DeltaMem gap disappears entirely: both matched summary memories reach 86.0% success, while strict DeltaMem remains at 78.0%. We therefore treat the main end-to-end comparison as policy-coupled and avoid stronger causal claims about the delta memory *unit* alone.

### A.11 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.12 Prompt templates (abridged but executable)

The artifact contains the exact prompt templates used in code; 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`,

| 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 3: Positioning DeltaMem-lite against representative memory-based agents. The table clarifies that our emphasis is on the *memory unit* and guarded reuse rather than on claiming a universally stronger end-to-end agent.

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

Table 4: Counterfactual help vs. harm on the currently evaluated DeltaMem-lite rerun subsets in the scale setting. Harm = original failure / no-patch rerun success. Help = original success / no-patch rerun failure. Net = Help - Harm.

| 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 5: Matched-policy pilot summary ($E$=50, seed 0). TRAJMEM and REFLMEM are rerun under the same top-1 + reuse-suppression policy as strict DeltaMem.

`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.13 Abstention dynamics (scale runs)

We measure conservatism at the step level by the rate of selecting a memory item *conditioned on memory candidates being present*. Table 6 shows that DeltaMem-lite abstains far more often than trajectory/reflection memories, consistent with guarded local patches.

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

Table 6: 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 7: 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).

### A.14 Conditioned memory cost (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 7 shows that DeltaMem-lite surfaces fewer candidates and injects fewer retrieved-memory tokens than trajectory/reflection memories.

In the same scale runs ($E$=100), DeltaMem-lite stores $32.0 \pm 3.3$items on ALFWorld and $22.0 \pm 5.7$items on WebShop (mean$\pm$std over 3 seeds), reflecting one-patch-per-failure writing with write gating and optional deduplication.

### A.15 Write gating and merging outcomes (scale runs)

To make the write policy transparent, Table 8 reports the number of candidate patches distilled (approximately the number of failures), and how many are added, rejected, or merged. At $E$=100, merges are rare; bank size is therefore primarily controlled by conservative write gating rather than deduplication.

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

Table 8: DeltaMem-lite write outcomes ($E$=100, mean±std over 3 seeds).

# B Additional Results and Analyses

## B.1 Scale 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.

## B.2 Delta patch taxonomy and qualitative examples

Figure 5 reports the distribution of patch levels in the final DeltaMem bank (pilot). Belief patches dominate the current pilot 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 9 provides representative delta items (abridged) highlighting minimal corrections with explicit anti-conditions.

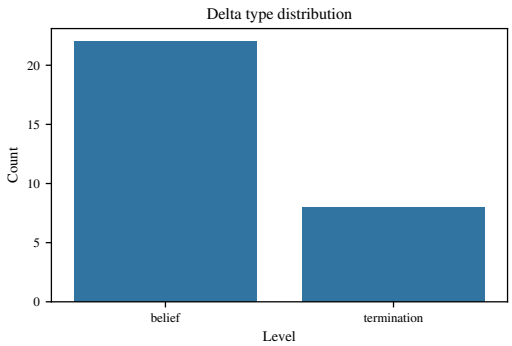


Figure 5: Distribution of delta levels in the final DeltaMem bank (pilot runs).

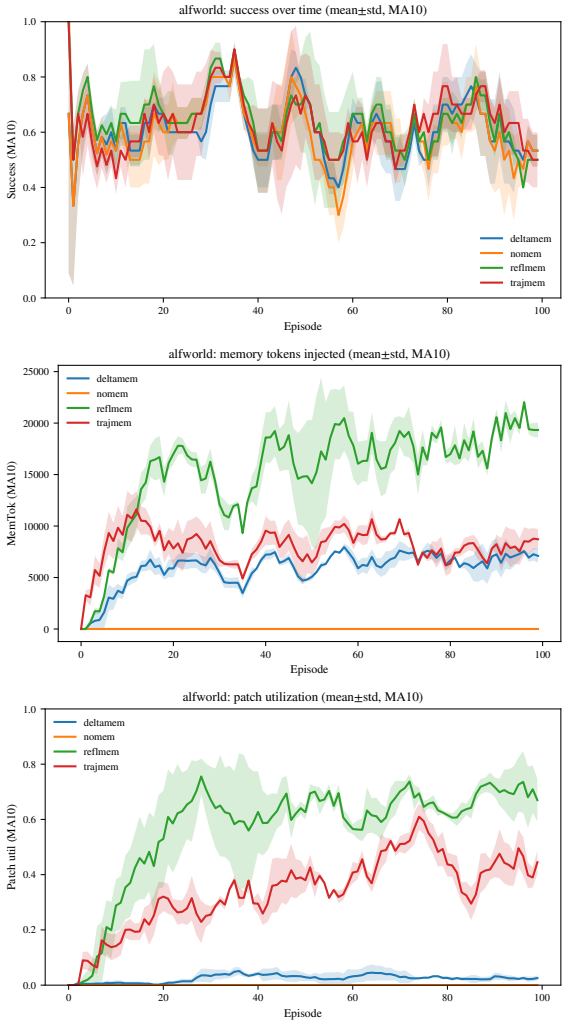


Figure 3: ALFWorld scale curves: success (top), retrieved-memory tokens (middle), and patch utilization (bottom).

| 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 9: Representative delta patches with triggers, minimal corrections, and explicit anti-conditions (abridged).

**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 sig-

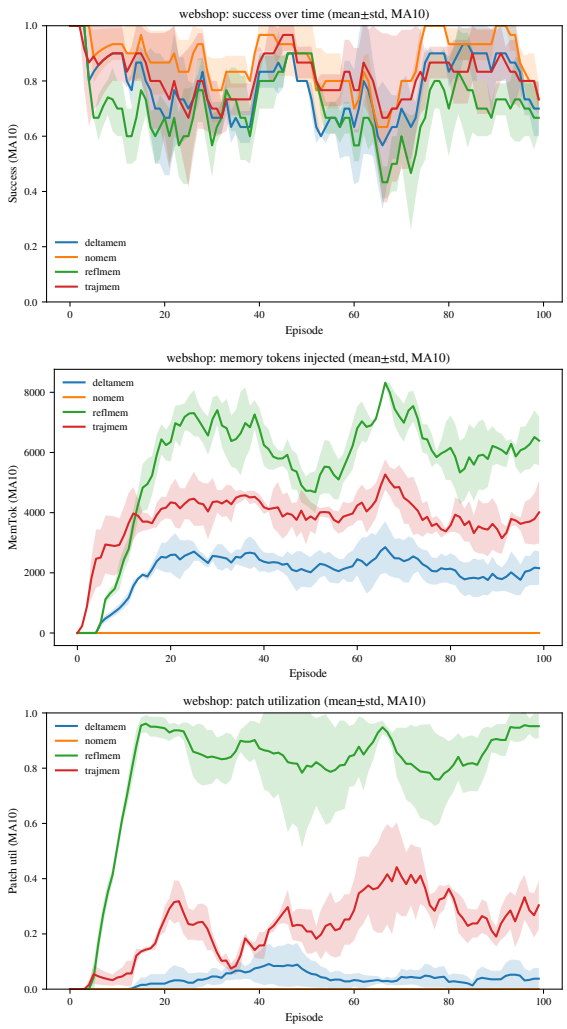


Figure 4: WebShop scale curves: success (top), retrieved-memory tokens (middle), and patch utilization (bottom).

nal appears. Figure 5 and Table 9 are consistent with this view: `belief` patches dominate the current 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.

### B.3 Pilot results and 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 10: Pilot results (E=50, seed 0) and DeltaMem design ablations.

### B.4 Conservative application ablations

Table 11 compares looser pilot 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 11: Looser pilot 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.