MORE MEMORY, WORSE AGENTS: ERROR REPRODUCTION AND ANTI-PERSISTENCE IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents can improve across tasks without updating model parameters by storing experience from their own executions in external memory and reusing it to guide later decisions. A central design question is whether to retain only observed trajectories and outcomes or also the rules, summaries, and skills inferred from them. Retaining these abstractions allows agents to reuse strategies without repeatedly deriving them from raw trajectories. However, errors in stored abstractions can propagate to subsequent decisions through retrieval and reuse. They can also shape execution trajectories, which then become the basis for new abstractions that reproduce the original errors. We term this feedback mechanism *error reproduction*: errors are regenerated through a cycle of abstraction, execution, and re-abstraction, extending their persistence beyond direct reuse of the original faulty knowledge. We propose ANTI-PERSISTENCE, a memory design that accumulates trajectories and outcomes while reconstructing task-specific guidance on demand. An adaptive selector uses prior outcomes to decide whether and how to synthesize guidance, which is discarded after execution. This prevents its direct reuse as a stored object, although indirect feedback through retained trajectories remains possible. Across different benchmarks and backbones, ANTI-PERSISTENCE achieves higher overall held-out success and the lowest average LLM-call cost among the evaluated memory designs. These results support separating experience accumulation from abstraction persistence in memory-based self-improvement.

1 INTRODUCTION

Large language model (LLM) agents solve tasks through multi-step interactions with an environment, producing trajectories of actions and observations (Yao et al., 2022b). We study *memory-based self-improvement*, in which an agent stores experience and outcomes from its own executions in external memory, retrieves that memory to guide later tasks, and updates it with the resulting experience (Shinn et al., 2023; Zhao et al., 2024; Song et al., 2024; Wang et al., 2023). Together, repeated execution, memory updates, and reuse form a feedback loop across tasks (Xiong et al., 2025; Du, 2026): with the LLM parameters held fixed, the agent adapts through changes in the experience available for decision-making and in how that experience is used. Although this process is intended to improve subsequent performance, its effectiveness must be established empirically, because the content retained by the agent can affect both the next task and later rounds of adaptation.

Stored experience can take the form of observed interaction records, which preserve trajectories and task outcomes, or inferred abstractions, which express rules, skills, summaries, or heuristics derived from those records. Because long interaction histories contain both reusable procedures and episode-specific details, distilling them into compact guidance can make relevant experience easier to retrieve and use. Many agent systems therefore extract abstractions from past executions and retain them for subsequent tasks. This compression nevertheless requires judgments about which details matter and under what conditions a pattern generalizes; a rule that succeeds in one task may depend on conditions absent from another. The value of abstraction thus depends not only on identifying a useful pattern but also on reassessing its applicability when it is reused.

The effects of a mistaken abstraction can extend beyond the task in which it is retrieved and used. A stored rule can influence an agent's actions, and the resulting execution records may become inputs for synthesizing new abstractions in subsequent self-improvement rounds (Figure 1). Even when the

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

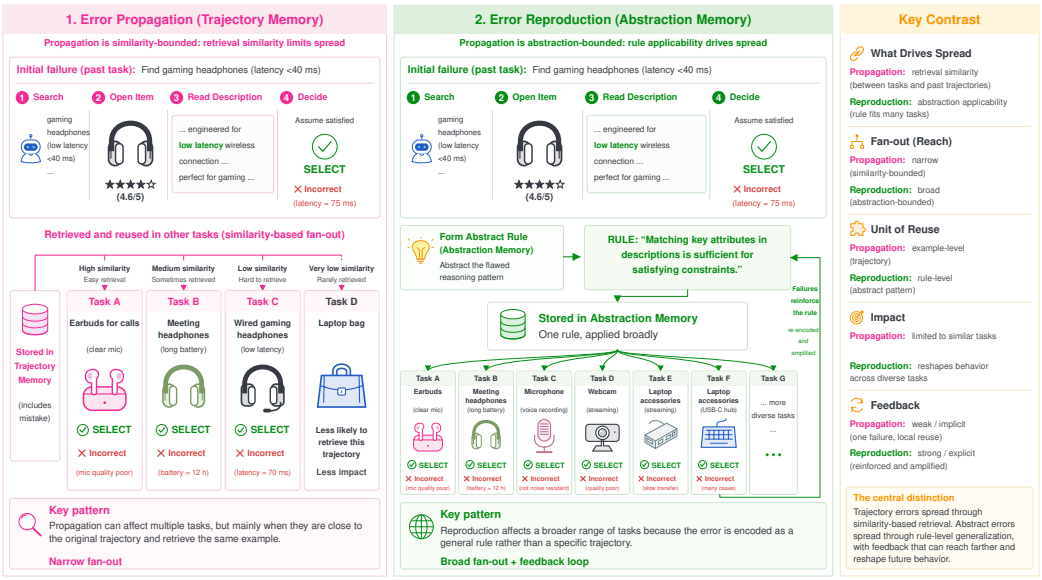


Figure 1: Error propagation and reproduction in agent memory. **Left:** An illustrative error in a stored trajectory is reused through retrieval. **Center:** A faulty inferred rule may shape later trajectories and be encoded again through subsequent synthesis. **Right:** Possible routes of influence. Both designs can retain feedback through stored trajectories; the examples imply neither a fixed ordering of their reach nor inevitable amplification.

original rule is not supplied to the synthesis step, these records may still contain erroneous actions taken under its guidance. When extracting lessons from these records, the agent may infer a new rule that reproduces the same error, allowing the error to persist indirectly through execution history. We call this possible feedback mechanism *error reproduction*, distinguishing the reappearance of an error in a newly generated abstraction from its propagation through retrieval and reuse. Both trajectories and abstractions can carry influence across tasks, with the extent of that influence depending on how the stored content is retrieved and used.

Memory maintenance can improve stored content through scoring, pruning, or refinement (Qiu et al., 2026), but deleting an abstraction does not by itself revise trajectories generated under its influence or remove abstractions subsequently derived from those trajectories. Whether reproduction continues depends jointly on which records are retrieved, what is synthesized from them, and which outputs are admitted to memory. These dependencies motivate examining the need for inferred content to persist alongside the procedures used to maintain it, leading to the following question:

How can LLM agents benefit from abstraction in memory-based self-improvement while limiting error reproduction?

We propose ANTI-PERSISTENCE, a memory module that retains observed interaction records while synthesizing abstractions for the current task. An adaptive selector uses prior outcomes to determine whether and how to transform retrieved experience into guidance, which supports the current execution and is discarded afterward. Subsequent tasks can revisit the accumulated records to construct guidance suited to their own context, allowing the agent to adapt through experience accumulation and mode selection while the synthesis procedure remains fixed and its outputs remain temporary. Our contributions are threefold.

- We introduce a memory design for self-improvement that separates the accumulation of execution experience from the temporary use of abstractions. Raw trajectories and task outcomes are retained, while inferred rules, summaries, and heuristics are used only for the current task and discarded afterward.
- We develop an adaptive mode selector for on-demand abstraction that chooses between using retrieved records directly and synthesizing behavioral patterns, task strategies, or corrective guidance for each task. This enables a fixed synthesis procedure to support successive tasks without persisting its outputs.

- We evaluate multiple rounds of self-improvement across three benchmarks and three backbones. ANTI-PERSISTENCE achieves a higher overall success rate on held-out tasks at a lower average LLM-call cost. We also find associations between earlier and subsequently generated abstractions, providing empirical evidence consistent with error reproduction.

2 RELATED WORK

Memory-based self-improvement in LLM agents. External memory allows agents to incorporate their own interaction experience into later decisions (Yao et al., 2022b; Song et al., 2024), with existing approaches differing in the content retained across updates. Trajectory-based methods store raw or lightly processed episodes for similarity-based retrieval (Xiong et al., 2025; Du, 2026), whereas other methods retain verbal reflections, cross-task rules, summaries, or executable skills (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2023; Chen et al., 2024; Yang et al., 2026; Zhang et al., 2026). Related forms of abstraction also appear in production agent systems (Zhang et al., 2025; Yang et al., 2024; Wang et al., 2024). We examine how the choice of persistent content affects later memory updates through the trajectories it helps generate.

Error propagation in iterative systems. Prior work examines how generated content can bias later behavior through repeated use (Xiong et al., 2025; Wyllie et al., 2024; Briesch et al., 2023), while related model-collapse studies show that repeated training on self-generated data can reduce distributional fidelity (Shumailov et al., 2024; Seddik et al., 2024; Gerstgrasser et al., 2024; Feng et al., 2024). Although parameter training differs from the external-memory updates studied here, these findings motivate examining feedback from generated experience. We focus on *error reproduction*, in which abstractions shape trajectories that inform subsequent synthesis: propagation concerns how an existing error affects behavior, whereas reproduction concerns whether that influence is encoded in a new abstraction. Because deleting the original abstraction does not by itself remove stored descendants, this dependence on earlier outputs creates a possible feedback loop across iterations (Perdomo et al., 2020; Pagan et al., 2023), even though maintenance can also target the descendant entries.

Memory maintenance and reliability. Related approaches improve memory or agent reliability through pruning, scoring, refinement, and learned operations (Qiu et al., 2026; Packer et al., 2023; Park et al., 2023; Madaan et al., 2023; Schick et al., 2023; Zhong et al., 2024). In systems that retain abstractions, these maintenance procedures determine which inferred content remains available for direct retrieval across tasks. ANTI-PERSISTENCE addresses the complementary question of persistence by synthesizing task-specific abstractions on demand and discarding them after use, while retaining the resulting trajectories for later retrieval and synthesis. This design removes direct abstraction reuse but still permits indirect feedback through experience.

3 ANTI-PERSISTENCE: A MEMORY DESIGN FOR SELF-IMPROVING AGENTS

ANTI-PERSISTENCE constructs temporary guidance from accumulated interaction records, combining on-demand synthesis with adaptive mode selection. We first formalize the memory update process and then describe how these components support task execution.

3.1 PRELIMINARIES

We consider an agent with fixed backbone parameters θ that processes a sequence of tasks, receiving task τ_k and producing trajectory e_k and outcome y_k at training-task update k. Each experimental iteration ℓ comprises multiple such updates followed by held-out evaluation with memory and related state frozen.

Memory. The agent maintains a trajectory memory $M_k = \{(\tau_j, e_j, y_j) : j < k\}$, which stores previous executions and their outcomes. An operator ϕ generates an abstraction from the available records: $s = \phi(H), \quad H \subseteq M$, where M is the trajectory store available at synthesis time, with task, mode, and decoding arguments omitted for clarity. The abstraction store S_k contains the summaries, rules, strategies, or skills currently retained; because maintenance may delete or transform entries, it need not include every abstraction generated previously.

Algorithm 1 Local mode selection and task execution under anti-persistence.

Require: Task τ; stores $M_k, \mathcal{L}_k$; modes $\mathcal{Z}$; budgets $B, \hat{B}$; exploration coefficient c; decay γ; cost weight λ. This branch assumes a nonempty trusted neighborhood, positive weight for every mode, and no warmup or exploration override.

Ensure: Trajectory $e_\tau^{(k)}$; outcome $y_\tau^{(k)}$; updated stores $(M_{k+1},\ \mathcal{L}_{k+1})$.

$\mathcal{N}_{\hat{B}}(\tau) \leftarrow R(\tau, \mathcal{L}_k, \hat{B})$ # retrieve outcome-log entries
Let $\mathcal{I}_{\hat{B}}(\tau) = \{1, \ldots, |\mathcal{N}_{\hat{B}}(\tau)|\}$ index $\mathcal{N}_{\hat{B}}(\tau)$ as $\{(\tau_i, \mu_i, y_i, r_i, t_i)\}_{i \in \mathcal{I}_{\hat{B}}(\tau)}$
$t_{\max} \leftarrow \max_{i \in \mathcal{I}_{\hat{B}}(\tau)} t_i$
for $i \in \mathcal{I}_{\hat{B}}(\tau)$ **do**
    $w_i \leftarrow \max(\mathrm{sim}(\tau_i, \tau), 0) \cdot \gamma^{t_{\max} - t_i}$ # local relevance weight
end for
for each mode $\mu \in \mathcal{Z}$ **do**
    $I_\mu \leftarrow \{i \in \mathcal{I}_{\hat{B}}(\tau) : \mu_i = \mu\}; \quad n_\mu \leftarrow |I_\mu|$
    $\hat{r}(\mu \mid \tau) \leftarrow \frac{\sum_{i \in I_\mu} w_i r_i}{\sum_{i \in I_\mu} w_i}$
    $U(\mu \mid \tau) \leftarrow \hat{r}(\mu \mid \tau) + c\sqrt{\frac{\log |\mathcal{N}_{\hat{B}}(\tau)|}{\max(n_\mu, 1)}}$
end for
$\mu^\star \leftarrow \arg\max_{\mu \in \mathcal{Z}} U(\mu \mid \tau)$ # adaptive mode selection
$H_\tau \leftarrow R_{\mu^\star}(\tau, M_k, B)$ # mode-specific retrieval (Section B.3)
$\sigma_\tau \leftarrow \phi(H_\tau, \mu^\star)$ # $\phi(H_\tau, \text{none}) = H_\tau$
$(e_\tau^{(k)}, y_\tau^{(k)}) \leftarrow \pi_\theta(\tau;\ C_\tau^{(k)} = \sigma_\tau)$
Let κ_τ be the realized extra synthesis-call count; $r_\tau^{(k)} \leftarrow \mathrm{clip}(y_\tau^{(k)} - \lambda\kappa_\tau, -1, 1)$
$M_{k+1} \leftarrow M_k \cup \{(\tau, e_\tau^{(k)}, y_\tau^{(k)})\}$
$\mathcal{L}_{k+1} \leftarrow \mathcal{L}_k$; append $(\tau, \mu^\star, y_\tau^{(k)}, r_\tau^{(k)}, t_{\text{new}})$ if eligible (Section B.5); **discard** σ_τ

Retrieval and generation. Given a new task τ_k, the agent retrieves a context $C_k = R(\tau_k, M_k \cup S_k, B)$, where R is a retrieval operator and B is the retrieval budget. The agent then generates a trajectory according to $e_k \sim \pi_\theta(\cdot \mid \tau_k, C_k)$, where π_θ denotes the underlying policy, such as an LLM-based agent.

Memory update. After each task, the agent stores the task, execution trajectory, and outcome: $M_{k+1} = M_k \cup \{(\tau_k, e_k, y_k)\}$. Methods that retain abstractions also update a separate store S_k. Let $\mathcal{A}_k$ denote new candidates synthesized from available records. Then $S_{k+1} = \mathcal{U}_k(S_k, \mathcal{A}_k; M_{k+1})$. Here, $\mathcal{U}_k$ decides how to incorporate these candidates and maintain existing entries, with M_{k+1} available as evidence. It may simply append all candidates or apply filtering, pruning, or merging.

Anti-persistence updates M_k but keeps $S_k = \emptyset$: guidance generated for the current task is discarded after execution. The execution records remain available for later retrieval and synthesis.

Feedback through experience. A stored abstraction s_{parent} can enter the agent's context during task execution, which produces trajectory e_j. Once execution is complete, its record may be included in retrieved records H and used to synthesize a new abstraction $s_{\text{candidate}}$: $s_{\text{parent}} \longrightarrow e_j \longrightarrow H \longrightarrow s_{\text{candidate}}$. An error in the old abstraction may thus influence execution and reappear in newly synthesized guidance. Appendix A separately analyzes exposure to flagged abstractions, generation of flagged candidates, and their admission to persistent memory.

3.2 Algorithm Design

Anti-persistence separates the use of an abstraction from its storage for later reuse (Section 3.1). It retrieves interaction records, generates guidance when needed, and discards that guidance after execution.

Persistent abstraction: trajectory → abstraction → storage → reuse
Anti-persistence (ours): trajectory → storage → retrieval → on-demand abstraction

What persists. The agent retains two kinds of records. Trajectory memory M_k stores tasks, executions, and outcomes that can be retrieved as examples. The outcome log $\mathcal{L}_k$, defined below,

records mode choices and their feedback to support later choices. Neither store retains synthesized guidance, and the abstraction store S_k remains empty.

These records preserve what occurred, without certifying that the agent's actions or explanations were correct. Temporary guidance may affect a stored trajectory and thereby influence later synthesis. Discarding guidance therefore prevents its direct reuse as a stored object but leaves indirect feedback through trajectories possible.

How to use retrieved experience. For each task, the agent chooses whether to use interaction records directly or synthesize guidance from them. We define four modes:

$$\mathcal{Z} = \{\text{none},\ \text{imitation},\ \text{strategic},\ \text{corrective}\}. \tag{1}$$

The none mode supplies retrieved records directly, so it uses memory without an additional synthesis call. The $\mathrm{imitation}$ mode extracts reusable action patterns and their conditions from successful trajectories. The $\mathrm{strategic}$ mode uses successful trajectories to organize higher-level subgoals and plans. The $\mathrm{corrective}$ mode analyzes failed trajectories to generate avoidance guidance, using a successful trajectory as a contrast when available. Full definitions appear in Section B.1.

Feedback for mode selection. The agent uses past outcomes to assess which mode may suit the current task. We formulate this choice as a lightweight contextual bandit over $\mathcal{Z}$, supported by the outcome log

$$\mathcal{L}_k = \{(\tau_j, \mu_j, y_j, r_j, t_j) : j < k,\ j \in \mathcal{J}_k\}, \tag{2}$$

where $\mathcal{J}_k$ indexes eligible completed attempts (Section B.5). Each entry records the task, selected mode, raw outcome y_j, cost-adjusted reward r_j, and logical timestamp t_j. The reward guides mode selection, while the raw outcome preserves task performance separately from the cost penalty.

Two retrieval steps. On the local-selection branch, the controller first retrieves similar tasks from $\mathcal{L}_k$ to choose a mode. It then retrieves trajectories from M_k according to that mode. The first retrieval supplies feedback for choosing how to use experience; the second supplies the experience itself. Their budgets are $\hat{B}$ and the mode-specific base budget B, respectively.

Selecting a mode from local feedback. For task τ, the first retrieval returns up to $\hat{B}$ nearest outcome-log entries, $\mathcal{N}_{\hat{B}}(\tau) = \{(\tau_i, \mu_i, y_i, r_i, t_i)\}_{i \in \mathcal{I}_{\hat{B}}(\tau)}$. The controller compares modes using their recorded rewards, weighted by relevance and recency.

Cost-adjusted reward. Each attempt receives a reward that combines task outcome with additional synthesis cost, $r_i = \mathrm{clip}(y_i - \lambda\kappa_i, -1, 1)$, where κ_i counts actual extra synthesis calls, including zero when synthesis is skipped. The coefficient $\lambda \geq 0$ controls the penalty, which applies only to synthesis calls.

Relevance weighting. Each retrieved entry receives a weight based on task similarity and recency, $w_i = \max(\mathrm{sim}(\tau_i, \tau), 0) \cdot \gamma^{t_{\max} - t_i}$, where $t_{\max}$ is the latest timestamp in the retrieved neighborhood and $\gamma \in (0, 1]$ controls the weight of older feedback. Anchoring at $t_{\max}$ rescales all γ^{k-t_i} weights equally and leaves normalized estimates unchanged. Decay affects only mode-selection feedback, without changing trajectory retrieval. An older mode evaluation may become less representative as memory grows, even when its trajectory remains a useful example.

Local utility estimation. For each mode, the controller averages the rewards of retrieved attempts that used it. If their total weight is positive, $\sum_{i:\mu_i=\mu} w_i > 0$, the estimate is

$$\hat{r}(\mu \mid \tau) = \frac{\sum_{i \in \mathcal{I}_{\hat{B}}(\tau):\, \mu_i = \mu} w_i r_i}{\sum_{i \in \mathcal{I}_{\hat{B}}(\tau):\, \mu_i = \mu} w_i}.$$

This estimate summarizes the mode's observed performance and synthesis cost on similar tasks.

Mode selection. The controller adds an exploration bonus to this estimate and selects the highest-scoring mode:

$$\mu^\star = \arg\max_{\mu \in \mathcal{Z}} \left(\hat{r}(\mu \mid \tau) + c \sqrt{\frac{\log |\mathcal{N}_{\hat{B}}(\tau)|}{\max(n_\mu, 1)}} \right), \tag{3}$$

Table 1: Held-out task success rate (%, mean ± std over 3 seeds) on ALFWorld, WebShop, and ScienceWorld with three backbones. Anti-persistence is compared with agent baselines and persistent-memory variants.

	Llama-4-Scout			Qwen3-Coder-Next			Gemma-4-31B-it		
Method	ALF	Web	Sci	ALF	Web	Sci	ALF	Web	Sci
ReAct	29.3±3.1	33.3±1.2	9.4±1.0	70.4±1.8	36.0±2.0	27.2±2.6	72.7±1.2	43.3±2.3	44.2±1.2
Reflexion	46.7±6.1	45.3±4.2	22.8±1.0	87.3±3.1	47.3±1.2	33.9±3.5	**98.0±0.8**	50.0±2.0	44.5±2.6
ExpeL	7.2±3.2	38.0±5.3	5.0±1.9	88.0±2.0	47.3±2.3	32.8±4.8	96.7±2.3	38.6±1.9	51.3±1.2
AutoRefine	36.3±3.2	37.3±2.3	15.3±1.7	83.7±6.7	40.7±2.3	27.8±1.0	88.0±4.0	48.0±4.0	43.3±0.0
Traj. Mem.	41.6±1.7	45.3±1.2	20.0±1.7	84.4±2.6	50.7±1.2	43.8±3.7	96.0±1.0	48.0±0.2	51.7±1.7
Abs. Mem.	42.7±1.2	37.3±1.2	18.9±1.0	84.7±2.4	38.3±3.5	34.0±1.0	92.7±2.3	45.3±4.2	46.7±1.7
Joint Mem.	33.3±2.3	35.3±1.2	7.2±1.9	81.9±0.2	47.0±1.7	36.3±0.6	95.3±3.1	51.3±1.2	**56.3±2.4**
Anti-pers.	**54.7±2.9**	**46.3±1.8**	**30.6±1.0**	**96.0±1.4**	**54.0±2.0**	**52.6±1.5**	97.3±1.2	**53.3±3.1**	54.1±6.3

where $n_\mu = \left|\{i \in \mathcal{I}_{\hat{B}}(\tau) : \mu_i = \mu\}\right|$. Here, n_μ counts retrieved attempts using mode μ. For $c > 0$, the bonus encourages revisiting modes with fewer local observations. This UCB-inspired rule balances observed reward with exploration. It carries no confidence-bound or regret guarantee for weighted, nonstationary observations. Algorithm 1 describes the local branch with a nonempty trusted neighborhood and positive weight for every compared mode. Unsupported means are undefined; initialization, fallback, and exploration overrides use separate branches (Section B.4).

Execution and feedback. After selection, the agent retrieves mode-appropriate trajectories, uses them directly or synthesizes temporary guidance, and executes the task. During training, execution records enter M_k and eligible feedback enters $\mathcal{L}_k$. Guidance is discarded after use; evaluation leaves both stores unchanged. New records change subsequent estimates and choices. The backbone, mode definitions, synthesis prompts, retrieval procedures, and selection rule remain fixed.

4 Experiments

We evaluate held-out success and cost (Section 4.2), performance across memory updates (Section 4.3), and individual design choices (Section 4.4). Provenance associations (Section E.3) and controlled content and retention studies (Section F) provide complementary evidence about the role of inferred guidance.

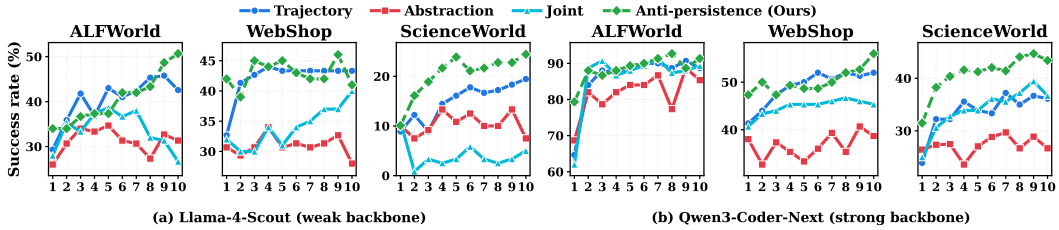
4.1 Experimental Setup

Benchmarks. We evaluate anti-persistence on ALFWorld (Shridhar et al., 2020), WebShop (Yao et al., 2022a), and ScienceWorld (Wang et al., 2022), using fixed training and held-out task partitions across methods and iterations following recent agent-memory work (Zhao et al., 2024; Chen et al., 2024; Shinn et al., 2023; Du, 2026). Partition sizes are reported in Table 4.

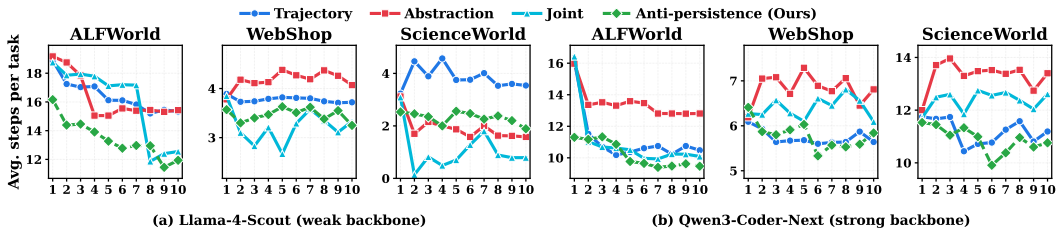
Backbones. The main experiments use Llama-4-Scout (Meta AI, 2025), Qwen3-Coder-Next (Cao et al., 2026), and Gemma-4-31B-it (Google DeepMind, 2026), spanning distinct model families and capabilities to examine how memory effects vary across backbones.

Baselines. We compare four memory configurations: **trajectory memory** retrieves stored trajectories, **abstraction memory** retrieves stored inferred abstractions, **joint memory** retrieves both, and **anti-persistence** constructs temporary guidance from stored trajectories. We also evaluate four agent baselines: ReAct (Yao et al., 2022b), Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and AutoRefine (Qiu et al., 2026).

Protocol. Each method undergoes $K = 10$ iterations with fixed backbone parameters, with memory and related state updated exclusively from training episodes. After each iteration, we evaluate the full held-out set with this state frozen: retrieval is permitted, but stored trajectories, abstractions, outcome logs, retrieval metadata, maintenance scores, and pruning decisions remain unchanged, and evaluation logs never enter subsequent updates. Under a shared per-task context budget, the base trajectory-retrieval budget is $B = 5$, with one additional successful trajectory available as a contrast



(a) Per-iteration held-out task success rate.



(b) Per-iteration mean trajectory length.

Figure 2: Per-iteration performance of four memory configurations over ten memory-update iterations across three benchmarks and two backbones.

for corrective mode (Section B.3). Anti-persistence uses up to $\hat{B} = 10$ outcome-log entries for mode selection.

Metrics. We report held-out success over 3 seeds ({42,43,44}), with error bars denoting standard deviations across seeds, and measure cost as the mean number of synthesis and execution LLM calls per held-out task.

4.2 Held-Out Performance and LLM-Call Cost

Table 1 compares held-out success at $K = 10$ against four agent baselines and three persistent-memory variants. Figure 3 reports average per-task LLM-call cost for the four memory configurations.

Anti-persistence improves overall held-out success. Anti-persistence achieved the highest success rate in seven of the nine benchmark–backbone settings (Table 1). Averaged equally across settings, it reached 59.9%, compared with 53.5% for trajectory memory, 49.0% for abstraction memory, and 49.3% for joint memory. It achieved higher mean success than trajectory and abstraction memory in all nine settings, and joint memory in eight. It also exceeded Reflexion, the strongest agent baseline on average, which reached 52.9%. The memory variants further clarify the role of stored abstractions: both abstraction memory and joint memory underperformed trajectory memory in seven of the nine settings. Retaining inferred guidance, either alone or alongside execution records, therefore did not consistently improve success. These comparisons favor anti-persistence overall while leaving the preferred design setting-dependent. Section 4.3 examines how these differences develop across updates.

Anti-persistence has the lowest average LLM-call cost. Across the nine settings, anti-persistence achieved the lowest mean LLM-call cost among the four memory configurations, alongside the highest mean success (Figure 3). Generating temporary guidance therefore did not translate into a higher average call count. Total call usage reflects both guidance synthesis and task execution. The none mode reuses retrieved trajectories directly without a synthesis call, while shorter execution trajectories in some settings offset calls used to construct guidance. These results support the efficiency of direct experience reuse and selective synthesis.

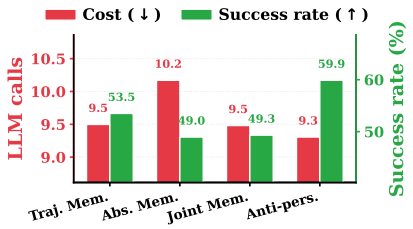


Figure 3: Mean LLM-call cost and held-out success across nine settings.

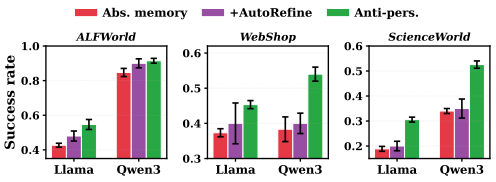


Figure 4: Final held-out success for abstraction memory, AutoRefine, and anti-persistence from Table 1. Bars show means over 3 seeds; error bars show standard deviations.

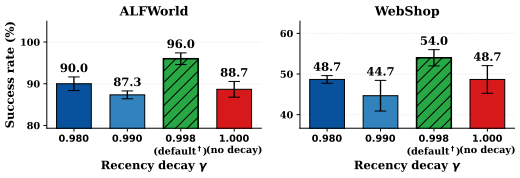


Figure 5: Sensitivity to recency decay γ on Qwen3-Coder-Next. Bars show means over 3 seeds; error bars show standard deviation across seeds.

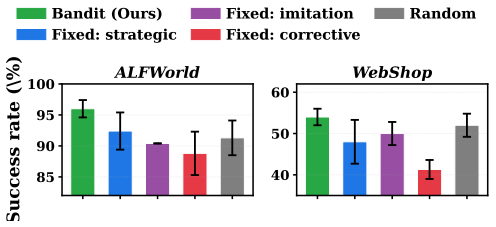


Figure 6: Mode-selection ablation on Qwen3-Coder-Next, comparing anti-persistence's contextual bandit with fixed-mode and random policies.

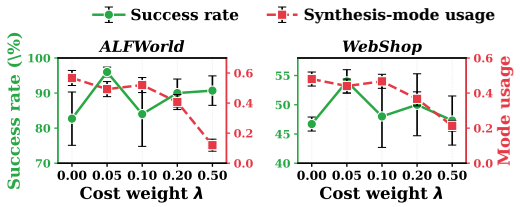


Figure 7: Sensitivity to λ on Qwen3-Coder-Next. Solid curves use the left axis for success rate; dashed curves use the right axis for synthesis-mode usage (fraction of tasks selecting a synthesis mode).

4.3 Dynamics Across Self-Improvement Rounds

To characterize behavior across memory updates, we first examine how the four memory configurations perform over successive rounds (Figure 2). We then compare the final results of abstraction memory, AutoRefine, and anti-persistence (Figure 4).

Memory Behavior Across Iterations. Memory effects varied across backbones and benchmarks (Figure 2). Anti-persistence achieved the highest final mean success in all six settings, although it did not lead at every iteration. Qwen's abstraction-memory performance fluctuated more on WebShop than on ALFWorld, while Llama's joint-memory performance declined in later rounds, particularly on ScienceWorld. These trajectories show that combining trajectory and abstraction memory did not consistently improve performance. Trajectory length provides complementary context (Figure 2b), but shorter rollouts can reflect either efficient execution or early failure. Their coincidence with lower success under joint memory is consistent with premature termination, without establishing its cause.

Comparison with Memory Maintenance. Figure 4 presents the final results for abstraction memory, AutoRefine, and anti-persistence from Table 1. AutoRefine improved mean success over abstraction memory in one of the six settings, matched it in one, and performed worse in four. Anti-persistence achieved higher mean success than AutoRefine in all six. AutoRefine also changes retrieval and synthesis (Section D.3), so this comparison does not isolate maintenance effects or rule out more effective policies. Deleting an abstraction leaves stored trajectories unchanged unless they are explicitly revised (Remark 1). Maintenance can nevertheless affect subsequent exposure, flagged generation, and persistent admission (Proposition 1).

4.4 Ablations and Design Analysis

Sensitivity to Recency Decay γ. The recency factor γ controls how quickly older outcome-log feedback loses weight during mode selection. Smaller values discount older feedback more strongly, whereas $\gamma = 1$ removes temporal decay while retaining task-similarity weighting. Figure 5 compares four values on Qwen3-Coder-Next. The default $\gamma = 0.998$ achieved the highest mean success among the tested values: 96.0% on ALFWorld and 54.0% on WebShop. At $\gamma = 0.980$, success was 90.0% on ALFWorld and 48.7% on WebShop; at $\gamma = 0.990$, it was 87.3% and 44.7%, respectively. Without decay ($\gamma = 1$), success was 88.7% and 48.7%. Both faster decay and no decay therefore underperformed the default. These results favor retaining historical feedback while modestly emphasizing recent outcomes, although success did not vary monotonically with γ.

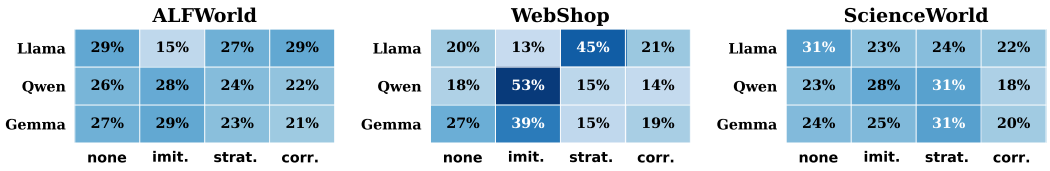


Figure 8: Mode-selection frequencies averaged over iterations 1–10. Panels denote benchmarks, rows denote backbones, and columns denote modes (none, imitation, strategic, corrective). Darker cells indicate more frequent selection.

Benefits of Adaptive Mode Selection. To assess the value of adapting mode selection to tasks and feedback, Figure 6 compares the contextual bandit with fixed strategic, imitation, and corrective policies and with random selection on Qwen3-Coder-Next. The bandit achieved the highest success rate on both ALFWorld (96.0%) and WebShop (54.0%). The strongest fixed policy differed by benchmark: strategic performed best on ALFWorld and imitation on WebShop. The bandit's advantage over fixed and random selection supports using task context and accumulated feedback to decide how retrieved experience should guide execution.

Sensitivity to Cost Weight λ. The cost weight λ penalizes additional synthesis calls in the mode-selection reward; $\lambda = 0$ removes this penalty. Figure 7 compares success and synthesis-mode usage across cost weights. The default $\lambda = 0.05$ achieved the highest mean success among the tested values on both ALFWorld (96.0%) and WebShop (54.0%), consistent with Table 1. Removing the penalty increased synthesis-mode usage but reduced mean success relative to the default; stronger penalties ($\lambda \in \{0.20, 0.50\}$) reduced both. These results favor selective synthesis with a modest penalty in the tested settings. Usage is the fraction of tasks selecting a synthesis mode, even when missing records prevent an actual call.

Mode Selection Patterns Across Settings. To examine how the controller allocated modes across settings, Figure 8 reports selection frequencies averaged over iterations for nine (backbone $\times$ benchmark) combinations. All settings used multiple modes. On WebShop, stronger backbones favored imitation and Llama-4-Scout favored strategic, while ALFWorld and ScienceWorld showed more balanced usage. The none mode, which reuses retrieved trajectories without synthesis, had a nonzero mean frequency in every setting, so the controller sometimes reused experience directly. These averages describe overall allocation, without showing whether a mode was selected in every round or establishing policy convergence.

Persisting versus Discarding Abstractions. To assess abstraction persistence, we compared variants that used synthesized guidance for the current task when synthesis was invoked, but differed in whether it was subsequently discarded or stored for later retrieval. Discarding improved success in all four settings (Table 2), with larger gains for Llama-4-Scout and positive gains for Qwen3-Coder-Next. These comparisons favor using synthesized guidance temporarily, without storing it for later tasks, in the evaluated settings.

Table 2: Held-out success with abstractions persisted or discarded (%, mean $\pm$ s.d.; 3 seeds).

Model	Bench	Discard (Ours)	Persist	Δ
Llama-4-Scout	ALF	**54.7±2.9**	26.7±6.1	+28.0
Llama-4-Scout	Web	**46.3±1.8**	34.0±4.0	+12.3
Qwen3-Coder-Next	ALF	**96.0±1.4**	92.7±1.2	+3.3
Qwen3-Coder-Next	Web	**54.0±2.0**	47.3±3.1	+6.7

5 Conclusion

We introduced anti-persistence for memory-based self-improvement: agents accumulate trajectories and outcomes and adaptively construct temporary guidance with a fixed backbone and synthesis procedure. The design removes direct storage and reuse of synthesized abstractions, while indirect feedback through trajectories remains possible. Our analysis distinguishes exposure, flagged generation, and persistent admission without implying that all maintenance policies fail. Across three benchmarks and three backbones, anti-persistence achieved higher overall held-out success with fewer LLM calls on average. These findings support separating experience accumulation from abstraction persistence and studying when verification can justify selective retention.

AI USE STATEMENT

We used generative AI tools to polish the manuscript's language, visualize the reported experimental data, and create workflow diagrams. The authors take responsibility for the final content of this paper.

Ethics statement. This work adheres to the ICLR Code of Ethics. Our study involves no human subjects, user studies, or collection of personal data. All experiments use publicly available models and established public benchmarks (e.g., ALFWorld, WebShop, and ScienceWorld), used in accordance with their licenses.

Reproducibility statement. We will publicly release our code upon publication to facilitate reproducibility.

REFERENCES

Martin Briesch, Dominik Sobania, and Franz Rothlauf. Large language models suffer from their own output: An analysis of the self-consuming training loop. 2023.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, et al. Qwen3-coder-next technical report. *arXiv preprint arXiv:2603.00729*, 2026.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631, 2024.

Pengfei Du. Memory for autonomous llm agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

Yunzhen Feng, Elvis Dohmatob, Pu Yang, Francois Charton, and Julia Kempe. Beyond model collapse: Scaling up with synthesized data requires reinforcement. In *ICML 2024 workshop on theoretical foundations of foundation models*, 2024.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, et al. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data. *arXiv preprint arXiv:2404.01413*, 2024.

Google DeepMind. Gemma 4: Byte for byte, the most capable open models. `https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/`, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`, 2025.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Nicolò Pagan, Joachim Baumann, Ezzat Elokda, Giulia De Pasquale, Saverio Bolognani, and Anikó Hannák. A classification of feedback loops and their relation to biases in automated decision-making systems. In *Proceedings of the 3rd ACM Conference on Equity and Access in Algorithms, Mechanisms, and Optimization*, pp. 1–14, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Juan Perdomo, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt. Performative prediction. In *International Conference on Machine Learning*, pp. 7599–7609. PMLR, 2020.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *arXiv preprint arXiv:2601.22758*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Mohamed El Amine Seddik, Suei-Wen Chen, Soufiane Hayou, Pierre Youssef, and Merouane Debbah. How bad is training on synthetic data? a statistical analysis of language model collapse. *arXiv preprint arXiv:2404.05090*, 2024.

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

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

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

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

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Sierra Wyllie, Ilia Shumailov, and Nicolas Papernot. Fairness feedback loops: training on synthetic data amplifies bias. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pp. 2113–2147, 2024.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. *arXiv preprint arXiv:2505.16067*, 2025.

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

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, et al. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, October 2025. Anthropic Engineering blog.

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

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

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

CONTENTS OF THE APPENDIX

A Reproduction Dynamics in Abstraction Memory **14**
- A.1 Operational Bias Labels and Historical Provenance . . . 14
- A.2 Exposure, Generation, and Persistent Admission . . . 14
- A.3 What Deletion and Maintenance Do Establish . . . 15
- A.4 Implications for Anti-Persistence . . . 15

B Method Details **16**
- B.1 Abstraction Modes . . . 16
- B.2 Mode-Specific Prompts . . . 16
- B.3 Retrieval Similarity . . . 18
- B.4 Hyperparameters . . . 19
- B.5 Outcome-Log Schema and Update Protocol . . . 19

C Worked Example: A Walkthrough of On-Demand Synthesis **20**

D Experimental Setup Details **22**
- D.1 Per-Benchmark Task Counts and Partitioning . . . 22
- D.2 Backbone Configurations . . . 23
- D.3 AutoRefine Baseline Implementation . . . 23

E Extended Cross-Backbone Results **25**
- E.1 Per-Iteration Curves on Gemma-4-31B-it . . . 25
- E.2 Per-Task LLM-Call Cost . . . 25
- E.3 Provenance Associations and Operational Bias Flags . . . 25

F Additional Controlled and Statistical Analyses **27**
- F.1 Effects of Skill Content on AppWorld . . . 27
- F.2 Selective Skill Retention on WebShop . . . 27
- F.3 Setting-Level Consistency Across the Main Evaluation Grid . . . 27

G Sensitivity and Robustness Studies **28**
- G.1 Trajectory Retrieval Budget B Sensitivity . . . 28
- G.2 Outcome-Log Retrieval Budget $\hat{B}$ Sensitivity . . . 29

H Limitations **29**

A Reproduction Dynamics in Abstraction Memory

This appendix formalizes the relationship between historical exposure to stored abstractions and the generation and retention of flagged candidates by distinguishing provenance, operational bias labels, and admission to persistent memory. Each abstraction symbol denotes a unique, immutable object identifier to which labels and parent sets refer; rewriting or merging creates a new identifier. The results concern a single synthesis invocation under stated conditions and establish neither progressive improvement nor inevitable long-run degradation, nor do they imply that all maintenance policies fail.

A.1 Operational Bias Labels and Historical Provenance

Definition 1 (Operational bias labels)**.** *Fix an evaluation and labeling protocol $\mathcal{E}$. Let $\mathcal{S}_{\mathrm{lab}}$ be the abstractions with defined labels under that protocol, and let $b_{\mathcal{E}} : \mathcal{S}_{\mathrm{lab}} \to \{0, 1\}$. Define the flagged set*

$$\mathcal{B}_{\mathcal{E}} = \{s \in \mathcal{S}_{\mathrm{lab}} : b_{\mathcal{E}}(s) = 1\}.$$

An abstraction in $\mathcal{B}_{\mathcal{E}}$ is called operationally biased. Abstractions outside $\mathcal{S}_{\mathrm{lab}}$ are unclassified; absence of a flag does not certify that an abstraction is benign.

The evaluation protocol and resulting labels remain fixed within each theoretical comparison, so all probability statements below concern operational flags rather than an unobserved intrinsic error rate. Our empirical protocol assigns these labels using associations with downstream success (Section E.3), which do not identify the causal effect of an abstraction.

Historical provenance. For a stored execution record $m_j = (\tau_j, e_j, y_j)$, let $P(m_j)$ contain the identifiers of persistent abstractions retrieved into the agent's context when e_j was generated. These identifiers remain attached to the record for analysis even after the corresponding abstractions are deleted, documenting historical exposure without by themselves establishing behavioral influence.

For a synthesis input $H \subseteq M_k$, define its historical parent set and flagged-parent exposure event by

$$P(H) = \bigcup_{m \in H} P(m), \qquad C = \{P(H) \cap \mathcal{B}_{\mathcal{E}} \neq \emptyset\}.$$

Execution records and abstraction identifiers remain distinct types throughout this analysis, and every parent in $P(m_j)$ must have existed when the record was generated, so parent sets never depend on future trajectories. We call C an "exposed input" event without assuming that the input inherited any particular error from a parent.

For each synthesized candidate $s = \phi(H)$, record $P(s) = P(H)$ at creation. A directed edge $u \to s$ exists when $u \in P(s)$; descendants are objects reachable along these recorded synthesis edges, excluding the starting object itself.

A.2 Exposure, Generation, and Persistent Admission

Consider a single synthesis invocation conditional on a fixed post-maintenance state, with its probability law encompassing task and mode selection, input retrieval, and synthesis randomness. Writing the candidate output as $s = \phi(H)$ with other conditioning arguments suppressed, define

$$A = \{s \in \mathcal{B}_{\mathcal{E}}\}, \qquad W = \{s \text{ is admitted as the same object to persistent abstraction memory}\}.$$

Rewriting or merging a candidate creates a new object requiring its own label and admission analysis; all probabilities below refer to the same fixed state and probability law for the invocation.

Definition 2 (Conditional reproduction coefficient)**.** *Let $q = \Pr(C)$. If $q > 0$, define*

$$\alpha = \Pr(A \mid C).$$

If $\Pr(A \cap C) > 0$, also define the conditional admission probability

$$\eta = \Pr(W \mid A \cap C).$$

The coefficient α is the flagged-output rate conditional on flagged-parent exposure. It is not an offspring count, a causal effect, or a growth factor.

These quantities may vary with the state, retrieval policy, synthesis operator, and maintenance procedure. A positive α alone establishes neither an increase in the flagged-output rate relative to other inputs nor the accumulation of errors across successive iterations.

A.3 What Deletion and Maintenance Do Establish

Remark 1 (Scope of parent deletion). Immediately before a deletion-only operation, let S^- be the abstraction store and $D^- \subseteq S^-$ its currently retained descendants of a parent s_0. Descendants are identified by the historical provenance relation above, possibly over multiple generations, and exclude s_0 itself. If the operation deletes $R \subseteq S^-$ and does not alter execution records or their provenance, then

$$S^+ = S^- \setminus R, \qquad D^+ = D^- \setminus R, \qquad M^+ = M^-.$$

In particular, deleting only s_0 leaves D^- unchanged, whereas deleting descendants can remove them. Removing a parent does not automatically erase its recorded history, but the remaining influence need not be irreversible.

These identities follow directly from the specified deletion operation, without implying that descendants survive other maintenance steps or that retained records are necessarily retrieved again.

Proposition 1 (One-step generation and retention of flagged abstractions)**.** *Under the common law specified above, suppose $q > 0$. Then*

$$\Pr(A \cap C) = q\alpha, \qquad \Pr(A) \geq q\alpha.$$

If $q\alpha > 0$, then

$$\Pr(A \cap C \cap W) = q\alpha\eta, \qquad \Pr(A \cap W) \geq q\alpha\eta.$$

Thus, positive q, α, and η imply a positive probability of generating and admitting a flagged candidate with flagged-parent exposure.

Proof. Since $q > 0$, the definition of conditional probability gives $\Pr(A \cap C) = \Pr(C)\Pr(A \mid C) = q\alpha$. The inclusion $A \cap C \subseteq A$ gives the first inequality. If $q\alpha > 0$, conditioning on $A \cap C$ yields $\Pr(A \cap C \cap W) = \Pr(A \cap C)\Pr(W \mid A \cap C) = q\alpha\eta$. The inclusion $A \cap C \cap W \subseteq A \cap W$ gives the second inequality. No independence assumption is required. □

When $q = 0$, flagged-parent exposure has zero probability and α is undefined; when $q > 0$ but $\alpha = 0$, generating a flagged candidate with flagged-parent exposure has zero probability and η is undefined. Neither case justifies assigning a value to the undefined conditional quantity.

Scope of the result. Proposition 1 provides a probability decomposition rather than a positive lower bound valid across all maintenance policies, because maintenance may change future exposure q, the conditional input or synthesis distribution determining α, and the admission probability η. For example, excluding all exposed records sets $q = 0$, while rejecting every flagged candidate prevents flagged candidates from entering persistent memory. By separating historical exposure, flagged generation, and persistent admission within a single synthesis invocation, the proposition identifies distinct components of the process without establishing inevitable amplification or a long-horizon performance ordering.

A.4 Implications for Anti-Persistence

Because anti-persistence admits no synthesized candidate to the abstraction store, W is impossible and $\Pr(A \cap W) = 0$ for every task-time synthesis invocation. This storage property does not establish $\Pr(A) = 0$, since temporary abstractions may still be flagged and can influence stored trajectories, subsequent synthesis, and mode selection.

The historical exposure event C tracks previously stored abstractions and therefore excludes feedback arising from temporary guidance. In a fresh anti-persistence run with no stored ancestors, $q = 0$ and the corresponding α is undefined, not zero. Both raw trajectories and abstractions can carry the effects of past decisions into later tasks, with the extent of this influence determined by retrieval and use rather than memory type alone. The structural guarantee is consequently limited to the absence of stored abstractions and their direct reuse, while the performance benefit remains an empirical finding.

B METHOD DETAILS

This appendix specifies the abstraction modes, prompts, retrieval procedures, and hyperparameters used by anti-persistence, together with the outcome log that connects observed task results to subsequent mode selection.

B.1 ABSTRACTION MODES

The mode set $\mathcal{Z} = \{\text{none}, \text{imitation}, \text{strategic}, \text{corrective}\}$ provides direct access to factual context or one of three forms of on-demand abstraction, as introduced in Section 3. While none passes retrieved records directly to the agent, imitation extracts reusable behavioral patterns, strategic develops higher-level task strategies, and corrective generates guidance for avoiding observed failures. When the required records are available, the three synthesis modes invoke ϕ at task time and discard its output after execution, using the input and output conventions in Section B.2.

$\mu = \text{none}$ **(direct factual context).** The agent receives factual context H_τ, comprising retrieved trajectory records and observed outcomes, directly without a synthesis call, so $\sigma_\tau = \phi(H_\tau, \text{none}) = H_\tau$ and $\kappa(\text{none}) = 0$. When no records are available, including at initialization or after an embedding failure, $H_\tau = \emptyset$ and the mode reduces to the underlying ReAct policy.

$\mu = \text{imitation}$ **(behavioral-pattern abstraction).** ϕ extracts reusable behavioral patterns from successful trajectories, including action–observation sequences, tool-use routines, and their applicability conditions, to provide task-specific guidance without copying a trajectory verbatim. This mode emphasizes reusable action patterns, whereas strategic focuses on higher-level sub-goal organization and planning. The resulting abstraction is injected under `## Behavioral Patterns (imitation)` and discarded after execution, with $\kappa(\text{imitation}) = 1$ when synthesis is invoked; if no successful trajectory is available, $\sigma_\tau = \emptyset$ and no synthesis call is charged.

$\mu = \text{strategic}$ **(multi-trace strategy distillation).** ϕ uses up to the top-B retrieved successful trajectories to generate a strategy covering sub-goal structure, tool-use patterns, and recovery heuristics, with a prompt that distinguishes transferable structure from instance-specific details. The resulting guidance is injected under `## Task-Specific Strategy (strategic reflection)`, incurring $\kappa(\text{strategic}) = 1$ when synthesis is invoked. If no successful trajectories are available, $\sigma_\tau = \emptyset$ and no call is charged.

$\mu = \text{corrective}$ **(failure-driven avoidance rules).** ϕ uses up to the top-B retrieved failure trajectories and one contrasting success when available, prompting an LLM to identify failure modes, critical decision points, and recurring error patterns. The output consists of avoidance rules of the form "do NOT do X because Y," optionally paired with corrective actions, and is injected under `## Avoidance Rules (corrective reflection)`. Invoking synthesis incurs $\kappa(\text{corrective}) = 1$.

Design choice. When no failure trajectories are available, ϕ returns $\sigma_\tau = \emptyset$ rather than falling back to strategic, preventing strategic guidance from being attributed to the corrective arm during bandit updates.

Cost convention. We define $\kappa(\mu) \geq 0$ as the number of additional LLM calls made by mode μ relative to direct trajectory reuse, so direct factual context has $\kappa(\text{none}) = 0$. Each synthesis mode (imitation, strategic, or corrective) incurs $\kappa(\mu) = 1$ when invoked and zero when synthesis is skipped because the required records are unavailable. The cost weight λ places these call counts on the reward scale used for success outcomes, with defaults reported in Section B.4.

B.2 MODE-SPECIFIC PROMPTS

Each synthesis mode in Section 3 uses a prompt to produce σ_τ and a wrapper to present the guidance under a mode-specific header in the agent's context. We provide the strategic and corrective templates, describe the imitation interface, and specify the wrappers below, including the direct factual-context wrapper used by none to bypass synthesis.

Strategic synthesis prompt. The strategic mode prompts the synthesis LLM to distill a transferable strategy from up to the top-B retrieved successful trajectories ($B = 5$ by default).

```
You are synthesizing a task-specific strategy for an agent.
The agent is about to attempt this task:
  "{instruction}"

Below are complete action-by-action trajectories from similar tasks
that SUCCEEDED. Each trajectory shows the exact tool calls made and
the observations received.

{success_text}
Analyze what the successful trajectories share: the overall approach,
the order of sub-goals, the tool usage patterns, the recovery
heuristics. Distinguish what is TRANSFERABLE (structure, sub-goal
ordering, tool patterns) from what is NOT (specific object names,
specific locations, specific observation text).

Output a concise strategy the agent should follow for the current
task. Include reasoning for each recommendation. Do NOT echo the
trajectories verbatim -- distill them into actionable guidance.
Output plain text only.
```

The placeholders `{instruction}` and `{success_text}` contain the current task description and up to the top-B retrieved successful trajectories, respectively, with trajectories concatenated as `-- Success` i `--` blocks. The synthesis LLM uses temperature 0 and a 1024-token budget.

Corrective synthesis prompt. The corrective mode prompts the synthesis LLM to extract avoidance rules from up to the top-B retrieved failure trajectories, using a single successful trajectory as a contrast when available.

```
You are synthesizing avoidance rules for an agent.

The agent is about to attempt this task:
  "{instruction}"

Below are trajectories from similar tasks that FAILED. Study them
carefully -- identify the failure modes, the decision points where
things went wrong, and the recurring mistakes:

{failure_text}

For contrast, here are a few SUCCESSFUL trajectories on similar
tasks. Use them sparingly -- only to show the correct move at the
points where failures went wrong:

{success_text}

Output concise avoidance rules ("do NOT do X because Y") and, where
helpful, the corrective action the successful trajectories took at
the same decision point. Include reasoning. Output plain text only.
```

The placeholders `{instruction}` and `{failure_text}` contain the current task description and up to the top-B retrieved failures, respectively. The placeholder `{success_text}` contains the top-1 successful trajectory when available and otherwise takes the value `(no successful reference available)`, allowing synthesis to proceed without a positive reference. Decoding settings match those of the strategic prompt.

Imitation synthesis interface. Given the current task and up to the top-B retrieved successful trajectories, imitation identifies successful action patterns, their supporting observations, and ways to adapt them to the current task without generalizing instance-specific object names or locations into rules. The synthesized guidance is presented through the abstraction wrapper below rather than as a verbatim copy of a reference trajectory.

Direct factual-context wrapper (none). The none mode supplies retrieved records and their observed outcomes without a synthesis call, using the following factual-context header:

```
## Retrieved Factual Context (none)
Below are records of related past task executions and their outcomes.
Use them as evidence; adapt to the actual observations of this task.

<retrieved trajectory records and observed outcomes>
```

Factual records retain their task instructions, observed outcomes, and action–observation sequences within the shared context budget, with formatting that adds no inferred behavioral rules and requires no synthesis LLM call.

Injection wrapper. The output of imitation, strategic, or corrective receives a mode-specific markdown header before being appended to the agent's system prompt.

```
## <mode-specific header>
The following was synthesized from similar past trajectories for
this specific task. Treat it as guidance -- adapt to the actual
observations you receive.

<synthesized abstraction>
```

The headers `Behavioral Patterns (imitation)`, `Task-Specific Strategy (strategic reflection)`, and `Avoidance Rules (corrective reflection)` identify the source mode alongside each synthesized abstraction. Under none, the agent receives the factual-context wrapper above, which is omitted only when no records are available.

B.3 Retrieval Similarity

Both retrieval pathways operate on a fixed task embedding $\mathbf{e}_\tau \in \mathbb{R}^d$ using the cosine similarity function

$$\mathrm{sim}(\tau_i, \tau) = \frac{\langle \mathbf{e}_{\tau_i}, \mathbf{e}_\tau \rangle}{\|\mathbf{e}_{\tau_i}\| \cdot \|\mathbf{e}_\tau\| + \epsilon}, \quad \epsilon = 10^{-12},$$

but serve distinct purposes: *trajectory retrieval* selects records for direct use or synthesis, whereas *mode-selection retrieval* selects outcome-log neighbors for estimating local bandit rewards. These purposes lead to differences in candidate sets, time decay, and task-type filtering.

Task embedding. For each task τ, we compute one embedding by concatenating the task instruction with a truncated prefix of the initial observation:

$$\texttt{embed_text} = \texttt{instruction} \,\|\, \texttt{"Initial state:"} \,\|\, \texttt{truncate}(\texttt{obs}, 500).$$

Including the initial observation helps distinguish tasks that share a goal template but differ in their starting state. Qwen3-Embedding-8B (**?**) computes this embedding in a single API call; if the call fails, $\mathbf{e}_\tau$ remains unset and the controller defaults to $\mu = \text{none}$ for the current task without an in-iteration retry.

Trajectory retrieval (input to ϕ). Let R_μ denote retrieval from trajectory memory under the filtering rules and budgets for mode μ. Under none, $R(\tau, M_k, B)$ returns up to B relevant factual records and their outcomes without synthesis or success-only filtering. The imitation and strategic modes select up to the top-B successes ($y = 1$), whereas corrective selects up to the top-B failures ($y = 0$) and one contrasting success when available. Each candidate set is ranked by task similarity after task-type filtering, with a default budget of $B = 5$ (Section B.4).

Mode-selection retrieval (input to the bandit). The contextual bandit in Section 3 searches the entire outcome log $\mathcal{L}_k$ using the same task embedding and cosine similarity, computing $\mathrm{sim}(\tau_i, \tau)$ for every entry in a single matrix–vector product. The top-$\hat{B}$ entries, or all entries if fewer are available, form the local neighborhood $\mathcal{N}_{\hat{B}}(\tau)$ used for the cost-adjusted, recency-weighted reward estimates in Section 3. If average neighborhood similarity falls below ρ, the bandit falls back to a global-best policy that ignores the neighborhood; both ρ and $\hat{B}$ are reported in Section B.4.

B.4 HYPERPARAMETERS

Table 3 lists the method-level hyperparameters and default settings for anti-persistence, with sensitivity analyses varying only the specified parameters. Symbols follow the main text, supplemented by parameters governing bandit exploration, while Section D reports run-level settings such as iteration counts, episodes per iteration, and per-step budgets.

Table 3: Method hyperparameters and default values. Sensitivity analyses vary the specified parameters.

Parameter (symbol)	Default	Description
Bandit: mode selection		
KNN neighborhood size $\hat{B}$	10	top-$\hat{B}$ entries form $\mathcal{N}_{\hat{B}}(\tau)$
UCB coefficient c	0.5	exploration bonus base
Recency decay γ	0.998	weight $w_i \propto \gamma^{t_{\max}-t_i}$
KNN trust threshold ρ	0.7	fallback to global-best if $\overline{\text{sim}} < \rho$
Per-mode warmup W_{init}	3	each mode forced at least W_{init} times before KNN-UCB
Round-robin warmup iterations	3	first 3 iterations enforce round-robin selection
ε-greedy rate ε	0.1	probability of uniform-random mode selection
Reward clipping range	$[-1, 1]$	per-task bandit reward clipped to this range
Reward shaping		
Cost weight λ	0.05	$r = \text{clip}(y - \lambda\kappa, -1, 1)$
Trajectory retrieval (input to ϕ)		
Success top-k (B)	5	success trajectories retrieved per call
Failure top-k (B)	5	failure trajectories (corrective mode only)
Contrasting success top-k	1	success reference for corrective mode
Task-type filter	on	retrieval restricted to the current task type
LLM calls		
Synthesis max tokens	1024	for ϕ in imitation, strategic, and corrective
Agent max tokens	1024	per action / step
Embedding		
Embedding model	Qwen3-Embedding-8B	one call per task; see Section B.3
Init-observation prefix length	500 chars	characters of `obs` concatenated into `embed_text`

The following notes explain the design choices underlying the numerical defaults.

Round-robin and ε-greedy as safeguards. The first three experimental iterations use round-robin selection to provide initial mode coverage, subject to the logging exclusions in Section B.5. After warmup, an ε-greedy rate of 0.1 maintains exploration by giving every mode a nonzero chance of selection, complementing the initial coverage provided by round-robin selection.

KNN trust threshold $\rho = 0.7$. When the average cosine similarity within $\mathcal{N}_{\hat{B}}(\tau)$ falls below ρ, the bandit switches to its global-best fallback, reducing reliance on weak local matches without guaranteeing accurate global estimates. We set ρ to 0.7 using a small validation set and perform no further tuning.

Reward clipping. Per-task bandit rewards $r = \text{clip}(y - \lambda\kappa, -1, 1)$ are recorded in the outcome log $\mathcal{L}_k$ using the realized synthesis-call count κ, which is zero when synthesis is skipped. Although the defaults ($y \in \{0, 1\}$, $\lambda = 0.05$, $\kappa \in \{0, 1\}$) already produce rewards within this range, clipping accommodates potential changes in the reward or cost scale.

B.5 OUTCOME-LOG SCHEMA AND UPDATE PROTOCOL

The outcome log $\mathcal{L}_k$ introduced in Section 3 records observed mode choices and outcomes without storing synthesized abstractions. Its schema and update protocol are specified below; because the recorded outcomes inform later mode selection, this factual log still permits feedback across tasks.

Record schema. Each entry of $\mathcal{L}_k$ is a tuple of seven fields:

- `embedding` (`list[float]`): the task embedding $\mathbf{e}_\tau$ from Section B.3, used by the bandit's KNN lookup;
- `mode` (`str`): one of $\{\text{none}, \text{imitation}, \text{strategic}, \text{corrective}\}$;
- `reward` (`float`, clipped to $[-1, 1]$): the cost-adjusted bandit reward $r = \text{clip}(y - \lambda\kappa, -1, 1)$;
- `raw_success` (`bool`): the binary task outcome y, independent of λ;
- `task_type` (`str`): the task family (e.g., `alfworld`);
- `task_id` (`int`): the index of τ in its benchmark task pool;
- `instruction` (`str`, truncated to ≤ 500 characters): the task instruction text, retained for stratified analysis and debugging.

Storage format. $\mathcal{L}_k$ is stored as a single append-only JSONL file with one record per line, without rewriting, compaction, or deletion of entries. The example below displays one record with line breaks and abbreviated embedding values for readability:

```
{"embedding": [0.0123, -0.0456, ..., 0.0789],
 "mode": "strategic", "reward": 0.95, "raw_success": true,
 "task_type": "alfworld", "task_id": 7,
 "instruction": "put a clean mug on the coffee table"}
```

Update protocol. After each training-task attempt, the agent appends one record to $\mathcal{L}_k$, subject to three skip rules:

1. *The initial task update $k = 0$ is excluded.* At $k = 0$, trajectory memory M_k is empty and all modes reduce to baseline behavior. Logging this outcome would attribute baseline noise to the selected mode.
2. *Embedding failures are excluded.* If the embedding API call for τ fails, the record cannot enter KNN retrieval and is dropped.
3. *Infrastructure failures are excluded.* Records from abnormal terminations, such as environment exceptions, are skipped because their outcomes cannot be attributed to the selected mode.

These rules exclude attempts with empty initial memory, missing embeddings, or infrastructure failures, but do not remove task-selection effects or other confounding from the remaining observations.

Read protocol. On startup, the JSONL file is streamed to reload $\mathcal{L}_k$ in file order, using the following conventions for timestamp assignment and corrupted records.

Relative timestamps. Logical timestamps t_i follow log order without being stored in the file, and each appended entry receives a timestamp t_{new} later than those of existing entries. Reloading assigns consecutive integer ranks that preserve order while ignoring elapsed time and previous timestamp spacing. For a fixed loaded log, anchoring decay at the latest neighborhood timestamp $t_{\max}$ rather than the current update index rescales every weight by the same positive factor, leaving normalized reward estimates unchanged.

Tolerance to corrupted lines. Malformed lines, such as those caused by interrupted writes, are skipped with a logged warning while loading continues, preventing a single corrupted record from invalidating the log.

C Worked Example: A Walkthrough of On-Demand Synthesis

This appendix presents a constructed ALFWorld example at task update $k = 177$, after the three-iteration warmup, illustrating the supported local-selection branch in which the neighborhood passes the similarity threshold, every mode has positive weight, and no exploration override occurs. The records, weighted outcomes, and synthesized guidance illustrate the procedure rather than document a measured execution trace.

Illustrative Walkthrough of Anti-Persistence at Task Update k=177 (ALFWorld)

Incoming task (τ, illustrative identifier `example_heat_egg`): *"Your task is to: heat an egg and put it on the diningtable."* Initial state lists `countertop 1, microwave 1, diningtable 1, cabinet 1-3, drawer 1-2, fridge 1, ...`

Step 1 – Candidate records from M_k (B=5). Five candidate records illustrate the available experience (Section B.3), including three distinct failures on related "heat-and-place" tasks. This pool provides context for the example; mode-specific retrieval occurs after mode selection in Step 3.

• j=1 (sim 0.92, y=0, update 175): "heat a tomato and put it on the diningtable" — issued `heat tomato 1 with microwave 1` without first taking the tomato; the action had no effect because the object was not held.
• j=2 (sim 0.88, y=0, update 176): "heat a potato and put it on the diningtable" — issued `move potato 1 to diningtable 1` before any heating step.
• j=3 (sim 0.86, y=0, update 174): "heat a mug and put it on the countertop" — searched cabinets and drawers for the microwave instead of going directly; exceeded step budget.
• j=4 (sim 0.82, y=1, update 173): "heat an apple and put it on the diningtable" (contrasting success).
• j=5 (sim 0.75, y=1, update 172): "heat a bread and put it on the diningtable."

Step 2 – Top-$\hat{B}$ neighbors from $\mathcal{L}_k$ and per-mode estimates ($\hat{B}$=10). This illustration uses recency-weighted similarity weights $w_i = \max(\text{sim}(\tau_i, \tau), 0) \cdot \gamma^{t_{\max} - t_i}$ with γ=0.9 and cost weight λ=0.20. Experimental defaults are listed separately in Section B.4. In this constructed neighborhood, direct-use records have κ=0 and every synthesis-mode record has one realized synthesis call (κ=1). The corresponding per-mode estimates are:

• **none**: n_μ=2, weighted-mean $y = 0.10$, $\hat{r} = 0.10$.
• **imitation**: n_μ=4, weighted-mean $y = 0.30$, $\hat{r} = 0.30 - 0.20 = 0.10$.
• **strategic**: n_μ=2, weighted-mean $y = 0.50$, $\hat{r} = 0.50 - 0.20 = 0.30$.
• **corrective**: n_μ=2, weighted-mean $y = 0.85$, $\hat{r} = 0.85 - 0.20 = 0.65$.

Step 3 – UCB mode selection. Adding the exploration bonus $c\sqrt{\log|\mathcal{N}_{\hat{B}}|/n_\mu}$ with c=0.5:

• **none**: $0.10 + 0.54 = 0.64$.
• **imitation**: $0.10 + 0.38 = 0.48$.
• **strategic**: $0.30 + 0.54 = 0.84$.
• **corrective**: $0.65 + 0.54 = \mathbf{1.19}$ ($\mu^\star$; bonuses rounded to two decimals).

The controller selects $\mu^\star$ = corrective because it has the highest score, drawing on candidate records that provide three distinct failure modes for analysis. By comparison, imitation abstracts behavior from successful records without explicitly analyzing those failures.

Step 4 – On-demand abstraction $\sigma_\tau = \phi(H_\tau, \mu^\star)$. The selected corrective mode uses the top-3 failures (j=1, 2, 3) and contrasting success (j=4). The synthesis operator ϕ then makes one LLM call (κ(corrective)=1) to extract avoidance rules:

```
## Avoidance Rules (corrective reflection)
```

Rule 1. `Do NOT heat <object> unless you are holding it.`
From j=1: the heating action failed because the object was not held.
Corrective action: `take` the object and keep it in inventory until `heat`.

Rule 2. `Do NOT place <object> on diningtable before heating.`
From j=2: placement alone leaves the heating requirement unmet;
completing the goal would require additional actions.
Order of operations: heat first, then transport, then place.

Rule 3. `Do NOT search cabinets/drawers for microwave 1.`
From j=3: microwave 1 is listed in the initial observation;
use `go to microwave 1` directly.

Contrast (j=4, success): `take` → `go to microwave 1` → `heat with microwave 1` → `go to diningtable 1` → `move to diningtable 1`.

Step 5 – Illustrative execution under σ_τ. Under ALFWorld's action semantics, `heat` operates on a held object. The following constructed trajectory $e_\tau^{(177)}$ applies all three rules; bracketed observations are abbreviated illustrations:

```
  > go to countertop 1  [egg 1, knife 1]
> take egg 1 from countertop 1  (Rule 1: hold before heating)
> go to microwave 1  (Rule 3: go directly)
> heat egg 1 with microwave 1  [heated]
> go to diningtable 1  (Rule 2: place after heating)
> move egg 1 to diningtable 1  [success]
```

Illustrative outcome: $y_\tau^{(177)} = 1$ in 6 steps.

Step 6 – Memory update and discard. The trajectory record enters M_k, the mode and outcome enter $\mathcal{L}_k$, and the synthesized abstraction is discarded:

- $M_{178} \leftarrow M_{177} \cup \{(\tau,\, e_\tau^{(177)},\, 1)\}$.
- $\mathcal{L}_{178} \leftarrow \mathcal{L}_{177} \cup \{(\tau,\, \text{corrective},\, 1,\, 0.80,\, t_{\text{new}})\}$, with $r = 1 - 0.20$.
- σ_τ is discarded and S_k remains empty, preventing direct reuse of this abstraction as a stored memory object.

Reading the example. Local outcome estimates and cost-adjusted UCB scores determine the selected mode, after which corrective synthesis analyzes three failure patterns in the constructed records: heating without holding the object, premature placement, and inefficient search. A contrasting success helps formulate avoidance rules. The alternative modes emphasize reusable behavior from successful executions, higher-level planning, or direct factual context without synthesis. Although the temporary abstraction σ_τ is stored in neither $\mathcal{L}_k$ nor M_k, its effects remain in the recorded trajectory and may inform later synthesis, while the outcome log influences subsequent mode selection.

D Experimental Setup Details

This appendix details the task splits, training and evaluation protocol, backbone configurations, and AutoRefine implementation used in the experiments. All memory and controller updates use training experience; held-out outcomes remain separate from the update process.

D.1 Per-Benchmark Task Counts and Partitioning

For each benchmark, a fixed *training set* supplies interaction experience for memory and controller-state updates with backbone parameters held fixed, while a separate fixed *held-out set* is reserved for evaluation. Where available, we adopt community-standard splits from prior work (Shinn et al., 2023; Zhao et al., 2024; Chen et al., 2024; Fang et al., 2025), with task counts reported in Table 4 and the per-iteration sampling procedure described below.

Table 4: Per-benchmark training and held-out task counts used in this study.

Benchmark	Training tasks	Held-out tasks	Evaluation set
ALFWorld	180	134	134 unseen tasks (Shinn et al., 2023; Zhao et al., 2024)
WebShop	200	100	100 held-out instructions
ScienceWorld	150	60	2 held-out instances per task type

ALFWorld. ALFWorld (Shridhar et al., 2020) provides 3,553 training games and 134 unseen evaluation games across six categories: *Pick & Place*, *Examine in Light*, *Clean & Place*, *Heat & Place*, *Cool & Place*, and *Pick Two & Place*. We adopt the 134-task held-out split used in prior work (Shinn et al., 2023; Zhao et al., 2024) and draw 30 tasks per category to obtain a balanced training set of 180 tasks. Although AutoManual (Chen et al., 2024) uses smaller per-category splits, our setup follows the larger Reflexion/ExpeL evaluation protocol while keeping the training set manageable.

WebShop. WebShop (Yao et al., 2022a) provides 12,087 crowd-sourced shopping instructions paired with target product descriptions and structured attributes. We use a fixed 100-instruction held-out set. This evaluation size differs from the original benchmark's 500-instruction test split (Yao

et al., 2022a). We uniformly sample 200 training instructions from the remaining 11,987 using a fixed random seed to ensure a common training stream across methods.

ScienceWorld. ScienceWorld (Wang et al., 2022) provides 30 task types in elementary science, including thermodynamics, electricity, and biology, with multiple parameterized variations of each type. We construct a stratified partition by randomly selecting $N_{\text{train}} = 5$ training instances and $N_{\text{held}} = 2$ nonoverlapping held-out instances for each of the 30 types. The resulting 150 training tasks and 60 held-out tasks cover every task type, enabling evaluation on unseen variations within each type. Sampling once with a fixed random seed gives all methods the same partition, supporting controlled comparisons within this paper despite the difference from prior protocols.

Per-iteration training subset. At each experimental iteration ℓ, we sample $N_{\text{train/iter}} = 50$ training tasks uniformly without replacement and attempt them sequentially in an order determined by a fixed random seed. These episodes alone may update each method's memory and related state according to its update procedure, yielding 500 training episodes per benchmark across $K = 10$ iterations. All methods receive the same sample at iteration ℓ, controlling for training-task composition in comparisons of memory designs.

After each iteration's training updates, each method is evaluated on the full held-out set with memory and related state frozen. Held-out episodes use a separate log that is never carried forward and cannot update trajectory or abstraction memory, $\mathcal{L}_k$, retrieval metadata, maintenance scores, the skill evolver, or pruning. Each episode permits up to 50 environment steps, with attempts exceeding this budget terminated as failures ($y = 0$).

D.2 Backbone Configurations

We evaluate three open-weight backbones from distinct model families—Llama-4-Scout, Qwen3-Coder-Next, and Gemma-4-31B-it—running locally through OpenAI-compatible inference endpoints with the hardware and context lengths in Table 5. All methods share the same agent prompt and per-step output budget to control these factors when comparing memory configurations.

Table 5: Backbone configurations.

Model	Inference hardware	Context length
Llama-4-Scout (Meta AI, 2025)	2× NVIDIA H100	64K
Qwen3-Coder-Next (Cao et al., 2026)	4× NVIDIA H100	256K
Gemma-4-31B-it (Google DeepMind, 2026)	4× NVIDIA L40S	256K

Embedding model. Qwen3-Embedding-8B (**?**) produces task embeddings for trajectory retrieval (Section B.3) and KNN lookup through an OpenAI-compatible embeddings endpoint on the same cluster. Anti-persistence makes one embedding call per task using the same encoder with every backbone, ensuring a common embedding representation for outcome-log similarity comparisons across configurations.

D.3 AutoRefine Baseline Implementation

Our AutoRefine baseline adapts the maintenance procedure of Qiu et al. (2026), using the same retrieval and per-iteration episode budgets as the other methods (Section 4.1). The five components below include utility scoring, retrieval, pruning, synthesis, and the update loop. This baseline therefore differs from abstraction memory in more than maintenance alone. Its results are reported in Table 1 and Figure 4.

(i) Per-skill utility score. For each stored skill s, the metadata triple $(n_s^{\text{inj}}, n_s^{\text{succ}}, t_s^{\text{create}})$ records its prompt-inclusion count, the number of successful episodes among those inclusions, and its creation

iteration. We define skill utility as

$$\text{score}(s) \;=\; \underbrace{\frac{n_s^{\text{succ}}}{n_s^{\text{inj}} + \varepsilon}}_{\text{effectiveness}} \cdot \underbrace{\log\bigl(1 + n_s^{\text{inj}}\bigr)}_{\text{frequency}}, \tag{4}$$

with $\varepsilon = 10^{-3}$, so the frequency factor discounts skills supported by only one or two retrievals, while skills that have never appeared in a successful episode receive zero. This metadata is stored in a JSON file alongside the skill files.

(ii) Multi-query retrieval with similarity floor and MMR diversification. For each episode, the backbone rewrites the task description into three queries emphasizing *strategy*, *pitfall*, and *actions*, which join the original description to form four query strings. Each query is embedded with Qwen3-Embedding-8B (Section B.3), and a candidate skill's relevance is its maximum cosine similarity across the queries. After discarding candidates below the similarity threshold of 0.30, we re-rank the remainder using Maximal Marginal Relevance with diversity weight $\lambda_{\text{MMR}} = 0.7$, selecting at each step the skill that maximizes

$$\lambda_{\text{MMR}}\,\text{rel}(s) \;-\; (1 - \lambda_{\text{MMR}}) \max_{s' \in \mathcal{S}_{\text{sel}}} \text{sim}(s, s'), \tag{5}$$

and concatenating the top 5 entries into the prompt. Candidates are drawn from the general-skill set, the common-mistakes set, and the benchmark's task-type-specific skill set.

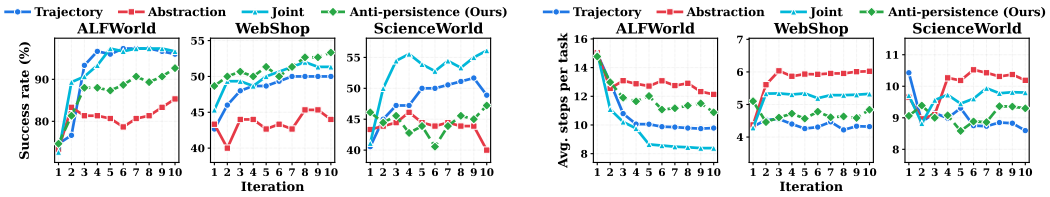
(iii) Periodic percentile pruning. Every two iterations (at $\ell = 2, 4, \ldots, K$), we prune the bottom 20% of eligible skills by utility score, with eligibility restricted to $n_s^{\text{inj}} \geq 3$ to protect skills with little retrieval history. A single transaction removes the selected entries from the in-memory dictionary, on-disk skill files, and metadata file while invalidating their embedding caches.

(iv) Contrastive skill evolver. After each iteration, a single contrastive prompt extracts new skills from at most four failure episodes and four sampled success episodes, using a benchmark-specific system preamble for ALFWorld, WebShop, or ScienceWorld. Each generated skill must include a `Why it works` section that contrasts the success and failure evidence. The synthesizer's output is decoded as a JSON array through a four-stage fallback parser comprising direct parsing, fenced-block stripping, extraction of the last bracketed array, and per-object scanning; skills with non-slug names are renamed `dyn-`N, using the smallest N above the maximum existing index. This single-call evolver replaces the original AutoRefine release's two-call procedure for reflecting on failures and extracting experience from successes, placing both forms of evidence in the shared context required by the contrastive prompt.

(v) Per-iteration update loop. Each AutoRefine iteration ℓ runs the following loop:

1. Reset the training retrieval log for the current iteration.
2. Run the training episodes for iteration ℓ, invoking the multi-query retrieval procedure in step (ii) for each episode.
3. Update metadata from the training retrieval log. For each skill included in an episode's prompt, set n_s^{inj} += 1 and, if that episode succeeded, n_s^{succ} += 1.
4. Run the contrastive evolver in step (iv) using only training outcomes. Add the new skills to the store and set $t_s^{\text{create}} = \ell$.
5. If $\ell > 0$ and ℓ is even, run percentile pruning (step (iii)) and write a metadata checkpoint.
6. Freeze the resulting state and evaluate on the full held-out set. Held-out episodes may retrieve skills, but a separate log records their retrievals and outcomes. These records do not update skill files, metadata, the evolver, or pruning.

The training retrieval log identifies each injection by the tuple (task type, task description, initial observation), and Step (3) credits a successful retrieval only when the skill appeared in that episode's prompt. Successes achieved without retrieving the skill therefore do not increase its utility score.



(a) Per-iteration held-out success rate.

(b) Per-iteration mean trajectory length.

Figure 9: Per-iteration performance on Gemma-4-31B-it across the three benchmarks.

Table 6: Average LLM-call cost per task after the final memory-update iteration (mean ± standard deviation over 3 seeds). Lower is better.

	Llama-4-Scout			Qwen3-Coder-Next			Gemma-4-31B-it		
Method	ALF	Web	Sci	ALF	Web	Sci	ALF	Web	Sci
Traj. Mem.	17.87±0.40	4.21±0.09	5.25±0.10	10.90±0.15	6.12±0.03	11.41±0.32	11.87±0.19	4.85±0.06	13.00±0.05
Abs. Mem.	16.74±4.77	4.53±0.27	1.33±1.15	13.64±0.03	6.91±0.10	15.08±1.05	13.46±1.04	6.04±0.17	13.80±0.59
Joint Mem.	19.11±0.48	3.23±1.57	1.15±0.86	10.95±0.33	7.21±0.10	13.93±0.95	11.01±0.50	5.96±0.17	12.78±0.71
Anti-pers.	15.13±0.19	3.99±0.05	3.56±0.22	11.84±0.28	6.64±0.55	12.68±0.40	11.20±0.38	5.30±0.19	13.41±0.19

E Extended Cross-Backbone Results

The main text reports per-iteration performance on Llama-4-Scout and Qwen3-Coder-Next (Figure 2) and aggregate per-task LLM-call cost (Figure 3). This appendix extends that evaluation with success-rate and trajectory-length curves for Gemma-4-31B-it (Section E.1), a cost breakdown for every (backbone × benchmark) setting (Section E.2), and an analysis of whether descendants with flagged parents are more likely to be flagged themselves (Section E.3). The latter describes historical associations relevant to the proposed error-reproduction pathway.

E.1 Per-Iteration Curves on Gemma-4-31B-it

Figure 9 reports held-out success rates (left) and mean trajectory lengths (right) across memory-update iterations for Gemma-4-31B-it on all three benchmarks. These results extend the comparison for Llama-4-Scout and Qwen3-Coder-Next in Figure 2.

Success rate. Among the four memory configurations, anti-persistence achieved the highest final mean success on ALFWorld (97.3%) and WebShop (53.3%). Joint memory performed best on ScienceWorld (56.3%, versus 54.1% for anti-persistence). These final values match Table 1, while the curves show that the relative ordering varied across earlier rounds.

Trajectory length. Joint memory produced the shortest trajectories on ALFWorld, while trajectory memory did so on WebShop, followed closely by anti-persistence. Trajectory memory also produced the shortest trajectories on ScienceWorld, where anti-persistence and joint memory had comparable lengths. Within each benchmark, the success-rate and trajectory-length panels cover the same iteration range for every method.

E.2 Per-Task LLM-Call Cost

Table 6 breaks down the aggregate cost in Figure 3 by reporting average per-task LLM-call counts for all four memory configurations across the nine (backbone × benchmark) settings. The metric includes all calls during task execution, including the additional synthesis calls under anti-persistence (Section B.1).

E.3 Provenance Associations and Operational Bias Flags

We describe an audit of parent–descendant associations relevant to the pathway in Section A, combining historical exposure records with outcome-based flags assigned after evaluation. The equations below specify the intended quantities and their data requirements, while Figure 10 retains the sup-

plied aggregates, which have not been recomputed for this revision or verified against event-level provenance and labeling-eligibility logs. The audit therefore remains descriptive and cannot establish a causal reproduction effect or a long-run growth coefficient.

Descendant tracking. For a candidate $s = \phi(H)$, its historical parent set is

$$P(s) = \bigcup_{m \in H} P(m),$$

where $P(m)$ records the persistent abstractions present in the context when execution record m was generated, retaining their historical identifiers even after the corresponding objects leave the current store S_k. Parent sets are recorded as analysis metadata when a candidate is created, without retroactively modifying the agent's memory. They document exposure and data lineage but do not establish that an abstraction caused a particular action or error.

Operational labeling. For each historically stored abstraction, the audit compares held-out episodes that retrieved it with episodes that did not. Let n_s^+, n_s^- and $\bar{y}_s^+, \bar{y}_s^-$ denote the respective episode counts and mean success rates. Define the set of abstractions eligible for labeling as

$$\widehat{\mathcal{S}}_{\text{lab}} = \{s : n_s^+ \geq n_{\min},\ n_s^- > 0\}.$$

For s in this set, assign

$$\widehat{b}(s) = \mathbf{1}\{\bar{y}_s^- - \bar{y}_s^+ \geq \delta\}, \qquad n_{\min} = 3, \quad \delta = 0.05,$$

and let $\widehat{\mathcal{B}} = \{s \in \widehat{\mathcal{S}}_{\text{lab}} : \widehat{b}(s) = 1\}$. Abstractions outside the eligible set remain unclassified throughout the audit, and both comparison groups must exist to avoid assigning labels from undefined means. These unadjusted operational flags instantiate Definition 1 without estimating causal harm, since retrieval may, for example, concentrate certain abstractions on more difficult tasks.

The audit uses held-out logs collected with memory and related state frozen, with labels and provenance analyses computed after the runs so that they cannot influence memory updates, maintenance scores, pruning, or subsequent behavior. Comparisons between the groups below require a common evaluation protocol and common audit thresholds.

Empirical conditional rates. Let $\mathcal{V}$ contain historically stored descendants that satisfy the labeling-eligibility conditions above and have a recorded historical parent set. Define $F(s) = \mathbf{1}\{P(s) \cap \widehat{\mathcal{B}} \neq \emptyset\}$. The intended descriptive group rates are

$$\widehat{\alpha}_{\text{emp}} = \frac{\sum_{s \in \mathcal{V}} F(s)\widehat{b}(s)}{\sum_{s \in \mathcal{V}} F(s)}, \qquad \widehat{\alpha}_0 = \frac{\sum_{s \in \mathcal{V}} (1 - F(s))\widehat{b}(s)}{\sum_{s \in \mathcal{V}} (1 - F(s))},$$

provided the corresponding denominator is positive, with a zero denominator leaving the rate undefined. The first group contains descendants with at least one flagged parent, whereas the second contains those with none; because the latter may include unclassified parents or an empty parent set, it does not constitute a group with verified benign ancestry.

These pooled observational rates describe conditional associations across the logged states rather than estimating α for a single fixed post-maintenance state. They also condition on prior storage and labeling eligibility, so they cannot separately identify candidate-level generation and admission rates. Comparing against $\widehat{\alpha}_0$ leaves task difficulty, retrieval selection, parent-set size, and other potential confounders uncontrolled, while shared ancestors and evaluation episodes introduce dependence among observations.

Results and scope. Descendants with at least one flagged parent had a higher flag rate in most of the six logged settings (Figure 10), with the largest reported gap, $\widehat{\alpha}_{\text{emp}} - \widehat{\alpha}_0 = +0.55$, occurring on Llama-4-Scout with ALFWorld. This association is consistent with the proposed pathway but identifies neither inherited error content nor a causal effect of parent exposure. The absence of group counts, labeling coverage, and an uncertainty-estimation procedure limits statistical interpretation of the aggregates, which cannot establish a causal amplification factor, a retention probability, or a threshold for long-run instability.

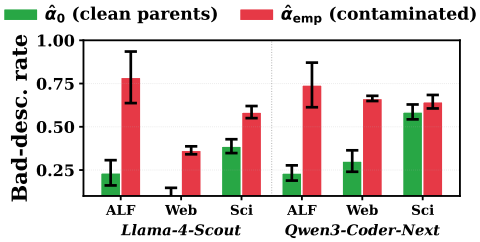
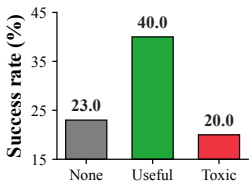
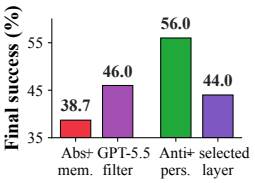


Figure 10: Observational flagged-descendant rates across six settings. Groups have at least one flagged parent or no flagged parent; absence of a flag does not establish benign ancestry. The largest reported rate gap is $+0.55$. Error bars are retained as reported, but their estimation procedure is unavailable with the aggregate data.



(a) AppWorld ($n = 30$).



(b) WebShop ($K = 10$).

Figure 11: Controlled abstraction-content and retention analyses; y-axes are truncated.

F Additional Controlled and Statistical Analyses

Three additional analyses examine the immediate effect of supplied skill content (Figure 11a), selective retention of filtered skills (Figure 11b), and consistency of performance differences across the nine main evaluation settings (Table 7). The two controlled analyses report aggregate success rates, with task-level and seed-level observations unavailable for estimating uncertainty.

F.1 Effects of Skill Content on AppWorld

We tested the immediate effect of skill content on 30 randomly selected AppWorld tasks (Trivedi et al., 2024), using matched pairs of reusable skills constructed by GPT-5.5 before evaluation. Each pair shared the same intended scope but contained a useful version with a correct procedural rule and a toxic version with an incorrect rule; these labels denote constructed conditions rather than the outcome-based flags in Section E.3. With the skill libraries frozen, Qwen3-Coder-Next attempted the same tasks under three conditions: no supplied skills, useful skills, and toxic skills.

With the acting model, task set, and intended skill scope held fixed, aggregate success was 23.0% without supplied skills, 40.0% with useful skills, and 20.0% with toxic skills (Figure 11a). These results characterize the immediate effects of supplied abstraction content on performance without establishing whether the resulting trajectories generate biased descendants or sustain error reproduction across iterations.

F.2 Selective Skill Retention on WebShop

We examined selective skill retention on WebShop with Qwen3-Coder-Next under two configurations: GPT-5.5 assessed each newly generated skill before admission to persistent abstraction memory in the first, while the selected skills were added to anti-persistence as a frozen persistent-memory layer in the second.

Within this auxiliary evaluation, filtering increased the final success rate of abstraction memory from 38.7% to 46.0%, whereas adding the frozen layer to anti-persistence reduced its final success rate from 56.0% to 44.0% (Figure 11b).

The tested filter and retention protocol thus improved the abstraction-memory baseline but reduced performance when the filtered skills were added to anti-persistence. Because the added layer remained frozen, its performance effect does not establish descendant generation or error reproduction across iterations.

F.3 Setting-Level Consistency Across the Main Evaluation Grid

To assess consistency across the nine fixed (backbone $\times$ benchmark) settings in Table 1, we computed the difference between anti-persistence and each baseline within every setting. We then summarized

Table 7: Exploratory comparisons across the nine fixed settings in Table 1. Δ denotes anti-persistence minus the named baseline, measured in percentage points. CIs are nominal, unadjusted 95% t-intervals, and all tests are two-sided. Holm correction covers the seven baseline comparisons within each test family. Shared backbones and benchmarks limit the inferential interpretation of these statistics.

Baseline	Mean Δ	95% CI	p_t	$p_{t,\mathrm{Holm}}$	p_W	$p_{W,\mathrm{Holm}}$	Positive cells
ReAct	$+19.2$	$[14.1, 24.4]$	< 0.0001	0.0002	0.0039	0.0273	$9/9$
Reflexion	$+\ 7.0$	$[2.7, 11.4]$	0.0060	0.0179	0.0078	0.0273	$8/9$
ExpeL	$+14.9$	$[3.6, 26.1]$	0.0157	0.0179	0.0039	0.0273	$9/9$
AutoRefine	$+13.2$	$[8.7, 17.6]$	0.0001	0.0007	0.0039	0.0273	$9/9$
Trajectory memory	$+\ 6.4$	$[2.8, 10.0]$	0.0036	0.0145	0.0039	0.0273	$9/9$
Abstraction memory	$+10.9$	$[7.6, 14.2]$	< 0.0001	0.0004	0.0039	0.0273	$9/9$
Joint memory	$+10.6$	$[3.6, 17.5]$	0.0079	0.0179	0.0195	0.0273	$8/9$

these paired differences by their equally weighted mean in percentage points and the number of settings with a positive difference.

For an exploratory statistical summary, we computed nominal 95% t-intervals, two-sided paired t-tests, and exact two-sided Wilcoxon signed-rank tests, applying Holm correction separately to the seven baseline comparisons within each test family while leaving the intervals unadjusted. All calculations used the one-decimal cell means in Table 1, with NumPy 2.5.0 and SciPy 1.18.1.

Anti-persistence achieved higher success in all nine settings against five baselines and in eight settings against Reflexion and joint memory (Table 7), with mean differences ranging from $+6.4$ to $+19.2$ percentage points. The direction of its advantage was therefore consistent across most or all settings for each baseline.

Although all seven Holm-adjusted p-values were below 0.05 in each test family, the shared backbones and benchmarks limit the treatment of the nine settings as independent observations. Holm correction addresses multiple comparisons without resolving this dependence, so the nominal intervals and p-values remain exploratory summaries rather than evidence for population-level significance. They neither quantify seed-level uncertainty within settings nor justify extrapolation to untested models and benchmarks.

G Sensitivity and Robustness Studies

We examine sensitivity to the two retrieval budgets introduced in Section B.4: the trajectory budget B, which controls the records supplied for direct reuse or synthesis (Section G.1), and the outcome-log budget $\hat{B}$, which controls the records used for mode selection (Section G.2).

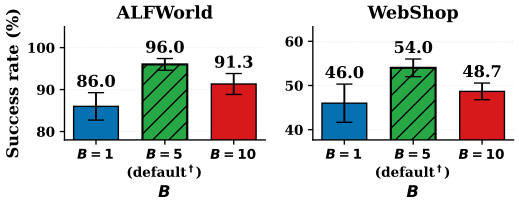
G.1 Trajectory Retrieval Budget B Sensitivity

The base trajectory retrieval budget B limits the records retrieved under the mode-specific rules (Section B.3): up to B records for direct reuse under none, up to B successes for imitation and strategic synthesis, or up to B failures and one optional success for corrective synthesis. To assess the effect of supplying fewer or more records as context, we evaluate $B \in \{1, 5, 10\}$ on Qwen3-Coder-Next with ALFWorld and WebShop, using three seeds per setting.

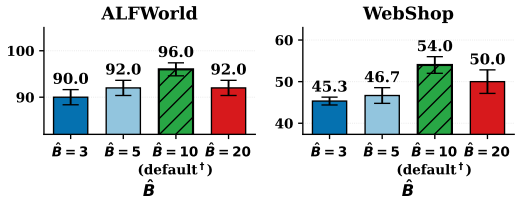
Observations. Figure 12a shows that both tested alternatives to the default retrieval budget yielded lower success rates.

(i) The smallest retrieval budget underperformed. At $B = 1$, direct reuse receives at most one record; synthesis receives at most one success for imitation or strategic mode, or one failure and an optional success for corrective mode. The resulting success rates of 86.0% on ALFWorld and 46.0% on WebShop were below the default, consistent with greater sensitivity to individual examples when only a few records are available.

(ii) The largest retrieval budget also underperformed. At $B = 10$, success rates were 91.3% on ALFWorld and 48.7% on WebShop, again below the default. Adding more weakly related trajectories



(a) Trajectory retrieval budget B. The default $B = 5$ yields the highest success rate among the tested values.



(b) Outcome-log retrieval budget $\hat{B}$. The default $\hat{B} = 10$ yields the highest success rate among the tested values.

Figure 12: Sensitivity to trajectory retrieval budget B and outcome-log retrieval budget $\hat{B}$ on Qwen3-Coder-Next. Bars show means over 3 seeds; error bars show standard deviations across seeds.

may dilute the useful context for direct reuse or synthesis, although this sweep does not isolate that explanation.

(iii) The default setting $B = 5$ performed best among the tested values. The default budget gave the highest observed success rates on both benchmarks, favoring an intermediate tested value as in the recency-decay analysis (Figure 5). This pattern is consistent with a trade-off between local relevance and coverage of retrieved experience.

G.2 Outcome-Log Retrieval Budget $\hat{B}$ Sensitivity

The outcome-log retrieval budget $\hat{B}$ limits the size of the bandit's local neighborhood and thus the number of nearby observations available for mode selection (Section B.4). We evaluate $\hat{B} \in \{3, 5, 20\}$ on Qwen3-Coder-Next with ALFWorld and WebShop, using three seeds per setting, and compare these alternatives with the default $\hat{B} = 10$ configuration in Table 1.

Observations. Figure 12b shows the same qualitative pattern as the trajectory-budget sweep, with the default neighborhood size outperforming both smaller and larger tested values.

(i) Smaller neighborhoods yielded lower success rates. With only a few observations available for local reward estimation at $\hat{B} = 3$, held-out success was 90.0% on ALFWorld and 45.3% on WebShop. Increasing the neighborhood to $\hat{B} = 5$ improved these rates to 92.0% and 46.7%, respectively, although both remained below the default, potentially reflecting greater sensitivity to individual outcomes in small neighborhoods.

(ii) The largest neighborhood also underperformed. At $\hat{B} = 20$, success rates were 92.0% on ALFWorld and 50.0% on WebShop, again below the default. Including more weakly related tasks may reduce the relevance of local reward estimates, although the sweep does not directly test this explanation.

(iii) The default setting $\hat{B} = 10$ performed best among the tested values. The default neighborhood size gave the highest observed success rates on both benchmarks, complementing the trajectory-budget sweep in Section G.1 and remaining consistent with a trade-off between local relevance and evidence coverage. These results support the default budgets within the tested ranges and settings without establishing a general optimum.

H Limitations

Anti-persistence prevents direct reuse of stored abstractions, but temporary guidance can still influence later synthesis through trajectories. Its performance also depends on retrieval quality and experience coverage. Our main iterative experiments cover three text-based benchmarks with fixed backbones and a fixed improvement procedure; broader generalization and longer-term gains remain untested. The two controlled auxiliary analyses lack task-level and seed-level uncertainty estimates, while the setting-level statistics remain exploratory (Section F). Discarding abstractions also forgoes the benefits of reusing reliable knowledge across tasks. Future work could examine when external verification or uncertainty estimates justify selective persistence.