MULTIFACETED GUIDANCE, TARGETED CORRECTION: EXPERIENCE HARNESS FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents can improve future behavior by reusing experience from past task executions. However, such reuse becomes challenging in realistic complex tasks involving sequentially dependent bottlenecks that differ in focus and are progressively revealed during execution. Consequently, generic reuse may overemphasize certain bottleneck focuses while overlooking others, or fail to provide timely intervention when errors emerge during execution. In this paper, we introduce Experience Harness, a specialized external control method that organizes experience from successful and failed trajectories and regulates how that experience guides and corrects an LLM agent. Specifically, it decomposes trajectories into bottleneck-aware atoms along four dimensions, inducing success skills for multifaceted task-start guidance and failure modes that trigger targeted corrective intervention during execution. This design preserves both the bottleneck focus of each experience and the stage at which it should take effect. Across multiple base agents on the AppWorld and BFCL-V3 benchmarks, our method consistently outperforms existing experience-memory baselines. Detailed experimental analyses further confirm the value of multifaceted guidance across bottleneck focuses and targeted correction at appropriate execution stages.

1 INTRODUCTION

LLM agents are used to solve complex tasks that require multi-step reasoning and tool-mediated interaction with external environments (Xie et al., 2024; Wölflein et al., 2025; Schick et al., 2023; Patil et al., 2024). Agent memory supports such tasks by preserving long-term user and environment context and retrieving relevant past observations during present interactions (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023). Beyond retaining past observations, agents should, like humans who revisit prior attempts to draw lessons for new situations, learn reusable knowledge from both successful and failed task executions. Prior work on experience memory pursues this goal by reflecting on trial outcomes (Shinn et al., 2023), distilling lessons from past tasks (Zhao et al., 2024), and inducing reusable workflow or procedural memories (Wang et al., 2025; Cao et al., 2025).

Despite this progress, extracting and reusing experience becomes more challenging as a complex task unfolds. Such tasks often involve sequentially dependent bottlenecks that differ in focus and become progressively observable during execution. In Figure 1, the agent must first recognize that the goal is to reimburse Mia rather than ask her for reimbursement; otherwise, every subsequent action is directed toward the wrong objective. This illustrates the **temporal dependency** between bottlenecks, as successful progress at later stages relies on earlier decisions being correct. Once the task intent is properly understood, execution reaches a different bottleneck at the tool-use stage: an incorrect parameter key is exposed only when the API call is attempted, illustrating how bottlenecks are **progressively revealed** during execution. After the correct reimbursement amount is obtained, the active bottleneck shifts again: the result must be delivered through the requested channel by emailing Mia. These bottlenecks are therefore **heterogeneous in focus**, spanning task understanding, tool use, and output construction. Consequently, the knowledge needed at each point is inherently **stage-contingent**: task-understanding knowledge matters at the outset, tool-use knowledge becomes critical during interaction, and delivery constraints become decisive near completion.

Prior experience-learning methods distill trajectories into reusable lessons, workflows, or procedural skills (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2025; Cao et al., 2025). However, they

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.

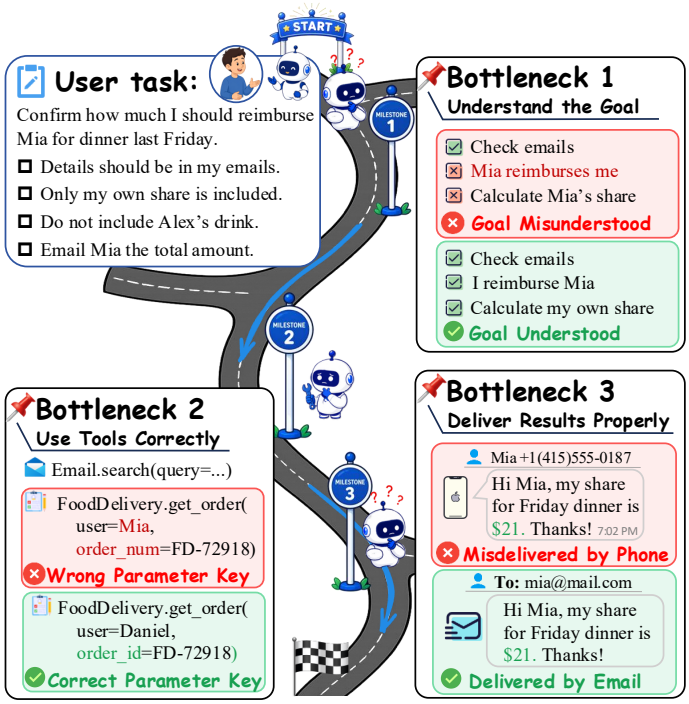


Figure 1: Illustration of heterogeneous, sequentially dependent bottlenecks in a complex agent task. Bottlenecks in goal understanding, tool use, and result delivery are progressively revealed as execution unfolds, with later progress depending on earlier decisions. Red and green panels contrast incorrect and correct handling at each stage.

do not explicitly unify these task properties within a stage-contingent framework for experience reuse. Recent work has begun to condition experience reuse on runtime context (Fan et al., 2026; Mi et al., 2026; Zhang et al., 2026). These advances address parts of runtime adaptation, but runtime context or uncertainty signals alone do not determine **intervention utility**: whether experience is genuinely needed, what role it should play and how it should be applied, and when its influence should end. Premature or persistent intervention may distract an agent that is already progressing correctly, whereas untargeted experience may fail to correct a local deviation before it propagates to later bottlenecks. Effective experience reuse must therefore account for not only contextual relevance, but also the role and lifecycle of experience during execution.

In this paper, we introduce **Experience Harness**, a specialized external control method that organizes experience derived from successful and failed trajectories. Instead of compressing each trajectory into a single lesson, we decompose trajectories into bottleneck-aware atoms along four complementary dimensions, each capturing a distinct bottleneck focus during task execution. These atoms are transformed into two complementary intervention mechanisms based on trajectory outcomes: success skills providing multifaceted task-start guidance and failure modes triggering targeted corrective intervention during execution. By separating guidance from correction, our method preserves the bottleneck focus of experience and the stage at which it should influence execution.

We conduct experiments on the AppWorld and BFCL-V3 benchmarks with multiple base agents. Compared with the strongest baseline, our approach improves the average task success rate (Avg@3) by 6.72% on AppWorld and 3.53% on BFCL-V3. Analyses further confirm the contribution of key components to the performance. These results show that structuring experience by bottleneck focus and regulating its role throughout execution can improve experience reuse in complex agent tasks.

Overall, our main contributions are outlined as follows:

- We characterize the challenges of experience reuse in complex agent tasks through heterogeneous, sequentially dependent, and progressively revealed bottlenecks, motivating experience organization by bottleneck focus and deployment by execution stage.

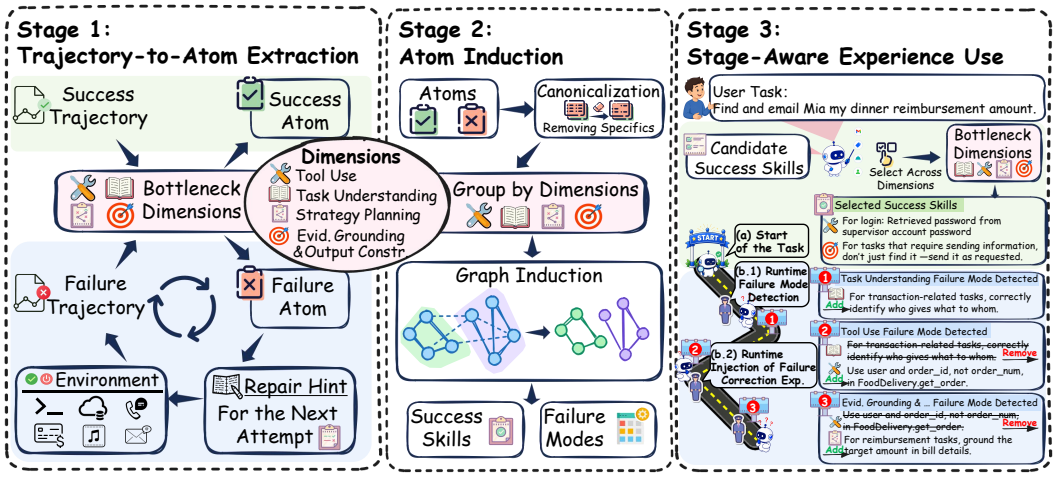


Figure 2: Method overview. Training trajectories are decomposed into bottleneck-aware success and failure atoms, induced into success skills and failure modes, and used at different test-time stages.

- We introduce experience harness to decompose trajectories into bottleneck-aware atoms and transform successful and failed experience into complementary mechanisms for multifaceted guidance and targeted correction.
- We assess our method on AppWorld and BFCL-V3 with multiple base agents. It achieves consistent improvements over prior memory and experience-learning baselines.

2 Related Work

Memory-enhanced LLM agents. Memory extends LLM agents beyond single-session reasoning by preserving observations, plans, and reflections across long-running interactions (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023), and cognitive architectures treat memory as a core substrate organizing perception, action, and decision making (Sumers et al., 2024). Recent work explores more structured forms: agentic memory organization (Xu et al., 2025), hierarchical memory for multi-agent systems (Zhang et al., 2025), workflow memory (Wang et al., 2025), and dynamic procedural memory for experience-driven evolution (Cao et al., 2025). These studies show that memory helps agents retain information, retrieve useful context, and reuse behavior patterns across tasks. We focus on how to extract useful experience and deliver it to the right execution stage: turning training trajectories into reusable guidance and correction for future tasks.

Experience learning and self-improvement for LLM agents. Experience learning improves LLM agents through interaction, feedback, and self-reflection. ReAct interleaves reasoning traces with actions (Yao et al., 2023), Self-Refine improves outputs via iterative self-feedback (Madaan et al., 2023), Reflexion stores verbal feedback from trial outcomes for future attempts (Shinn et al., 2023), and Voyager accumulates executable skills via open-ended exploration (Wang et al., 2024). Closer to our setting, ExpeL distills lessons from prior tasks (Zhao et al., 2024), while AWM and ReMe reuse workflow or procedural memories for future execution (Wang et al., 2025; Cao et al., 2025). These methods show that past interactions can improve future behavior; rather than treating experience as a single reusable lesson, we emphasize its distinct roles in guidance and correction during execution. Separately, MAST (Cemri et al., 2025) develops a 14-mode failure taxonomy for multi-agent systems through expert annotation of execution traces, serving post-hoc failure diagnosis. In contrast, our dimensions are not a failure ontology but an organization prior, and the concrete failure modes are induced automatically from data.

3 Preliminaries

Agent tasks and execution trajectories. We consider complex interactive tasks in which an agent solves a task through multi-step interaction with an environment. At each step t, the agent chooses an

action a_t from an action space $\mathcal{U}$, and the environment returns an observation o_{t+1}. The interaction history before step t is denoted by $h_t = (o_0, a_0, \ldots, a_{t-1}, o_t)$. Following the common ReAct-style formulation (Yao et al., 2023; Zhao et al., 2024), the agent follows a policy π that selects its next action based on the current history, optionally assisted by an external memory M:

$$a_t \in \mathcal{U}, \quad a_t \sim \pi(\cdot \mid h_t, M). \tag{1}$$

The new observation is appended to the history for subsequent decisions. This process continues until the task completes, fails, or reaches a step budget. An execution produces a trajectory:

$$\tau = (o_0, a_0, o_1, \ldots, a_T, o_{T+1}, y), \tag{2}$$

where y denotes the final outcome. Recording how agents interpret goals, explore solution paths, use tools, respond to feedback, and succeed or fail, trajectories provide a source of reusable experience.

Agent experience memory. Following prior work (Zhao et al., 2024; Wang et al., 2025; Cao et al., 2025), we define an experience memory $M = \{e_i\}_{i=1}^{n}$ as a collection of reusable natural-language knowledge extracted from past task executions to guide or correct future agent behavior.

Bottleneck dimensions. Norman's action cycle views the goal-directed behavior as a process of goal formation, action execution, and feedback interpretation (Weld, 1989). Inspired by this view, we organize trajectory-derived experience around four bottleneck dimensions,

$$\mathcal{Z} = \{z^{task}, z^{tool}, z^{plan}, z^{ground}\}, \tag{3}$$

where z^{task} denotes Task Understanding, z^{tool} denotes Tool Use, z^{plan} denotes Strategy Planning, and z^{ground} denotes Evidence Grounding & Output Construction. Each experience is associated with a dimension $z_i \in \mathcal{Z}$, indicating whether it mainly helps the agent understand the objective, execute tool actions, organize the workflow, or ground task-relevant information into the final answer. We note that these dimensions offer complementary views of task execution rather than an exhaustive coverage of all possible bottlenecks; within each dimension, concrete success skills and failure modes are induced automatically from data (Section 4). Detailed definitions of these dimensions are provided in Appendix D.

Extraction and evaluation. We study experience memory under a train-test evaluation protocol. Given a benchmark, let $D = D_{\text{train}} \cup D_{\text{test}}$, we construct experience memory from D_{train} trajectories and apply the fixed memory to tasks in D_{test}, without accessing test trajectories during construction.

4 METHODOLOGY

We design an experience harness method that transforms successful and failed training trajectories into complementary intervention mechanisms for multifaceted guidance and targeted correction, as illustrated in Figure 2. Given training trajectories $\mathcal{T}_{\text{train}} = \{\tau_i\}_{i=1}^{N}$, where each τ_i contains its final outcome y_i, our method has three stages. First, it extracts fine-grained success atoms $\mathcal{A}^+$ and failure atoms $\mathcal{A}^-$ from $\mathcal{T}_{\text{train}}$ along the bottleneck dimensions in Eq. (3). Then, it aggregates these atoms within each bottleneck dimension to induce two forms of reusable experience: success skills M^+ and failure modes M^-. Finally, it uses the two memories in a stage-aware manner at test time: success skills give task-start guidance while failure modes offer runtime correction.

4.1 TRAJECTORY-TO-ATOM EXTRACTION

Execution trajectories contain rich but heterogeneous learning signals. Summarizing an entire trajectory as a single experience may mix signals from different bottlenecks, emphasizing some while overlooking others. The resulting experience may also be either too task-specific to transfer or too coarse to retain actionable detail. We decompose trajectories into bottleneck-aware atoms along the dimensions in Eq. (3). An atom is a fine-grained, self-contained unit of local evidence that captures what contributes to success or failure at one bottleneck and why. Rather than representing an action or subgoal, an atom isolates the evidence associated with one bottleneck for cross-trajectory induction. These atoms are canonicalized and aggregated across trajectories to induce reusable experience.

Success atoms. Successful executions reveal what works. From each successful trajectory, an LLM extracts success atoms along the four bottleneck dimensions:

$$\mathcal{A}^+ = \bigcup_i \alpha_i^+ = \bigcup_i f_+(\tau_i, \mathcal{Z}), \text{ if } y_i = \text{success}. \tag{4}$$

Algorithm 1 Experience induction from atoms

Require: Atom set $\mathcal{A}$, dimension set $\mathcal{Z}$, thresholds λ_c, λ_s
Ensure: Induced experience memory M

```
M ← ∅
for all z ∈ Z do
    A_z ← {α ∈ A : dim(α) = z}
    Canonicalize all atoms in A_z
    for all α_i ∈ A_z do
        Retrieve a local neighbor set N(α_i)
        Use an LLM judge to identify mergeable groups in {α_i} ∪ N(α_i)
        Update pair counts c_ij and merge counts m_ij
    end for
    Build G_z = (V_z, E_z) where (α_i, α_j) ∈ E_z iff c_ij ≥ λ_c and m_ij/c_ij ≥ λ_s
    for all C ∈ CC(G_z) do
        e_C ← Synthesize(C)
        M ← M ∪ {e_C}
    end for
end for
return M
```

Each success atom α_i^+ records why the trajectory τ_i succeeds along one bottleneck dimension. Its schema is dimension-specific: workflow and tool-use atoms capture reusable steps, while task-understanding and grounding atoms capture principles, constraints, and evidence alignment.

Failure atoms. Unlike a successful execution, a failed attempt may terminate before downstream bottlenecks become observable. We collect failure atoms iteratively (Shinn et al., 2023; Madaan et al., 2023). At iteration r, the agent attempts task x and produces trajectory τ_r. We ask an LLM to identify the earliest critical deviation still obstructing task progress along the predefined bottleneck dimensions:

$$\alpha_r^- = f_-(\tau_r, \mathcal{Z}). \tag{5}$$

A failure atom α_r^- is a structured local record of this deviation and contains key fields: *locus* specifies the bottleneck dimension; *trigger context* describes the local decision context where the deviation occurs; *discrepancy* explains the gap between the actual and the expected move; *immediate consequence* records how the deviation affects task progress; and *repair hint* states how the agent should correct the deviation.

We use the *repair hint* of each candidate failure atom as a corrective experience e_r^- for the next attempt:

$$\tau_{r+1} = \mathrm{Run}(\pi, x, \{e_r^-\}), \quad e_r^- = g(\alpha_r^-), \tag{6}$$

where $\mathrm{Run}()$ executes policy π on task x using $\{e_r^-\}$, and $g()$ converts the repair hint in α_r^- into e_r^-.

After observing τ_{r+1}, we assess whether the repair contributes to task progress and collect validated failure atoms for later induction:

$$\mathcal{A}^- = \bigcup_{r:\gamma_r=1} \alpha_r^-, \quad q(\alpha_r^-, \tau_{r+1}) = \gamma_r. \tag{7}$$

where $q()$ denotes an LLM-based validation function and $\gamma_r \in \{0, 1\}$ indicates task-level contribution. If the task remains unsolved, the process repeats, allowing later deviations to surface.

4.2 Atom-to-Experience Induction

Individual atoms still reflect specific task executions and may not generalize on their own. We aggregate atoms that share common bottleneck patterns, inducing transferable success skills from repeated successful strategies and recurring failure modes from repeated deviations.

Canonicalization and grouping. We first canonicalize each atom by removing task-specific details such as names, IDs, exact values, and one-off entities while preserving its transferable boundary. The canonicalized atoms are then grouped by bottleneck dimension. This prevents, for example, tool-use patterns from being merged with task-understanding patterns merely because they come from superficially similar tasks.

Table 1: Main results on AppWorld and BFCL-V3. The rightmost Cross-Benchmark columns average the corresponding Avg@3 and Pass@3 scores across the two benchmarks.

Models	Methods	AppWorld			BFCL-V3		Cross-Benchmark	
		Avg@3	Pass@3	Score@3	Avg@3	Pass@3	Avg@3	Pass@3
Qwen3-32B	No Memory	25.59	41.66	60.59	42.78	48.33	34.19	44.99
	ExpeL	30.01	46.42	60.30	45.27	45.83	37.64	46.12
	AWM	26.98	45.83	59.96	45.83	49.17	36.40	47.50
	G-Memory	32.57	47.61	62.86	46.79	50.83	39.68	49.22
	ReMe	31.61	47.02	61.86	45.54	49.17	38.57	48.10
	MetaFlowLLM	29.96	45.23	59.64	45.27	49.17	37.61	47.20
	Skill-Pro	30.16	45.83	60.56	46.66	52.50	38.41	49.16
	Ours	**36.70**	**49.40**	**63.25**	**49.44**	**52.50**	**43.07**	**50.95**
Qwen3-235B-A22B-Instruct	No Memory	49.40	66.66	76.73	49.04	60.83	49.22	63.75
	ExpeL	52.10	67.85	79.91	55.55	63.33	53.83	65.59
	AWM	51.16	68.45	78.54	51.67	61.67	51.41	65.06
	G-Memory	55.13	69.64	80.54	53.89	64.16	54.51	66.90
	ReMe	54.06	67.26	78.65	53.01	62.50	53.53	64.88
	MetaFlowLLM	55.35	68.45	80.35	53.61	63.33	54.48	65.89
	Skill-Pro	56.15	**70.23**	81.90	54.44	64.16	55.29	67.19
	Ours	**58.13**	69.64	**82.61**	**57.50**	**65.83**	**57.82**	**67.74**
GLM-4.6	No Memory	60.70	70.23	81.42	74.83	78.33	67.76	74.28
	ExpeL	63.21	72.02	83.20	76.19	80.83	69.70	76.42
	AWM	61.12	70.83	82.10	75.27	79.16	68.19	74.99
	G-Memory	64.01	73.80	84.20	77.03	80.00	70.52	76.90
	ReMe	64.73	71.42	83.20	78.11	80.00	71.42	75.71
	MetaFlowLLM	65.07	73.21	84.80	75.83	79.16	70.45	76.18
	Skill-Pro	63.09	71.42	82.18	78.61	80.83	70.85	76.12
	Ours	**67.65**	**75.59**	**85.95**	**79.72**	**81.67**	**73.69**	**78.63**

Graph-based induction. For each dimension $z \in \mathcal{Z}$, we construct a merge graph $G_z = (V_z, E_z)$, where $V_z = \mathcal{A}_z$ is the set of canonicalized atoms assigned to dimension z. To avoid all-pair comparisons, we retrieve a local neighborhood for each atom and use an LLM to judge which atoms form the same reusable experience. Repeating this process yields pairwise support statistics. For atoms α_i and α_j, let c_{ij} denote the number of times that the two atoms appear in the same comparison window and m_{ij} denote the number of times that they are judged mergeable. We define the merge score as $s_{ij} = \frac{m_{ij}}{c_{ij}}$. We add an edge when the pair has enough comparison coverage and sufficiently high merge support:

$$(\alpha_i, \alpha_j) \in E_z \iff c_{ij} \geq \lambda_c \wedge s_{ij} \geq \lambda_s. \tag{8}$$

The connected components $\mathrm{CC}(G_z)$ serve as candidate clusters for experience induction. Success and failure atoms are processed separately using the same procedure (Algorithm 1).

The final synthesis depends on the atom type: success clusters are summarized into success skills, while failure clusters are summarized into failure modes. Success skills use dimension-specific schemata, with strategy and tool-use skills emphasizing steps and task-understanding and grounding skills emphasizing principles, constraints, and evidence alignment. Failure modes summarize the shared error mechanism, especially its working definition and discrepancy kernel, so that they can be detected and corrected during execution.

4.3 Stage-Aware Experience Use

Experience should not only guide the agent before execution, but also support exploration and correction during execution. At test time, our method realizes this by task-start success guidance and runtime failure correction.

Task-start success guidance. Before execution, we retrieve success skills from training tasks whose instructions are similar to the current one. For each bottleneck dimension, an LLM selects the skills that are relevant to the current task:

$$S_x = \mathrm{Select}_{\mathrm{LLM}}(\mathrm{Retrieve}(x, M^+), x, \mathcal{Z}). \tag{9}$$

The selected skills S_x are injected as lightweight, useful prior guidance before execution begins.

Runtime failure correction. During execution, the agent maintains a set of active failure modes. For every K steps, a runtime detector inspects the recent trajectory and determines whether the behavior matches any known failure mode:

$$F_t = \mathrm{Detect}_{\mathrm{LLM}}(h_t, M^-). \tag{10}$$

Thus, the memory injected at step t is $M_t = S_x \cup F_t$. If a mode is detected, the corresponding corrective experience is activated and injected into the agent context. The detector also checks whether active modes have been resolved. Once a failure mode is no longer blocking the task, it is deactivated and its experience is removed. This prevents a local correction from over-constraining the agent after the relevant bottleneck has been addressed. Together, task-start success guidance and runtime failure correction implement stage-aware experience use: success skills help the agent begin with a better task strategy, while failure modes intervene only on signs of a known deviation.

5 Experiments and Results

5.1 Experiment Settings

Datasets. We evaluate on AppWorld and BFCL-V3, two interactive tool-use benchmarks covering realistic APP/API environments and multi-turn function calling (Trivedi et al., 2024; Patil et al., 2025). We build memory from 90 AppWorld and 60 BFCL-V3 training tasks, and evaluate on 168 AppWorld Test Normal and 120 BFCL-V3 test tasks (Wang et al., 2026; Cao et al., 2025).

Metrics. Each task is executed three times. We report Avg@3, the task-averaged success rate over the three executions, and Pass@3, the proportion of tasks solved at least once. For AppWorld, we also report Score@3, the average AppWorld after-score over the three executions (Trivedi et al., 2024).

Models and baselines. We employ Qwen3-32B, Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025), and GLM-4.6 (GLM, 2025) as base models. We compare against a no-memory baseline and six methods using reusable experience: (1) ExpeL distills textual lessons from prior trials for future reuse (Zhao et al., 2024); (2) AWM constructs agent workflow memory to support later task execution (Wang et al., 2025); (3) G-Memory organizes hierarchical memory for subsequent reasoning and collaboration (Zhang et al., 2025); (4) ReMe builds dynamic procedural memory through experience-driven refinement (Cao et al., 2025); (5) MetaFlowLLM retrieves hierarchical MetaFlows generalized from completed trajectories to guide runtime execution through reusable workflows and dynamically completed subtasks (Fan et al., 2026); (6) Skill-Pro learns procedural skills from interaction experience and activates relevant skills during execution (Mi et al., 2026).

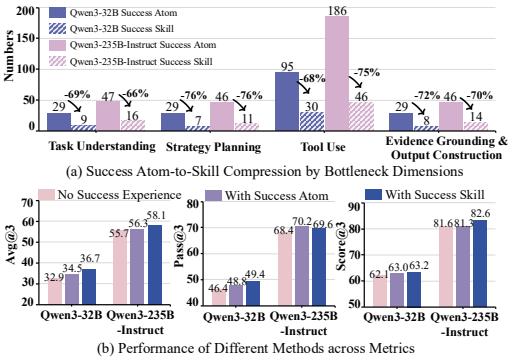
5.2 Main Results

Table 1 shows that our method achieves the highest Avg@3 across all base models on both benchmarks. Compared with the no-memory baseline, it achieves relative Cross-Benchmark Avg@3 gains of 25.97%, 17.47%, and 8.75% with Qwen3-32B, Qwen3-235B, and GLM-4.6, respectively. Pass@3 follows the same overall trend, indicating that our method improves both per-run success and the chance of solving a task at least once. For every combination of base model and benchmark, our method outperforms the best-performing prior baseline in Avg@3, which also provide experience-based guidance during execution, suggesting benefits beyond simply injecting experience at runtime. Complete results additionally including Kimi are reported in Appendix B

5.3 Analysis of Success Skills and Failure Modes

We analyze whether success skill induction produces a more useful guidance memory. As depicted in Figure 3(a), induction substantially condenses success atoms while preserving their bottleneck structure: 182 atoms are reduced to 54 skills for Qwen3-32B, and 325 atoms are reduced to 87 skills for Qwen3-235B. Tool Use contains the largest number of atoms and skills for both models, suggesting that tool-related experience is especially frequent in AppWorld.

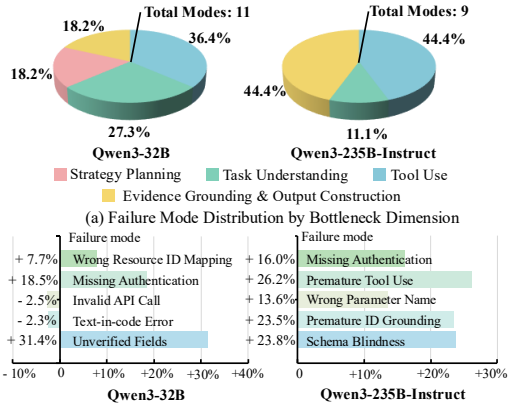
Figure 3(b) compares no success guidance, raw success atoms, and induced success skills. Raw atoms already improve over no success guidance, showing that successful trajectories contain useful



(a) Success Atom-to-Skill Compression by Bottleneck Dimensions

(b) Performance of Different Methods across Metrics

Figure 3: Analysis of success skills on AppWorld. (a) shows how success atoms are aggregated into a small set of success skills. (b) compares task performance when using no success guidance, raw success atoms, and induced success skills.



(a) Failure Mode Distribution by Bottleneck Dimension

(b) Net Improvement Rate for Top-5 Activated Failure Modes

Figure 4: Analysis of failure modes on AppWorld. (a) shows their distribution across bottleneck dimensions. (b) reports the net improvement rate of the top-5 most frequently activated failure modes.

signals. Induced skills improve Avg@3 by 2.18 over raw atoms with Qwen3-32B and 1.79 with Qwen3-235B, while also achieving the best Score@3 for both models. These results suggest that induction makes success-side experience more compact and effective as task-start guidance.

Next, we analyze whether induced failure modes form reusable corrective signals during execution. Figure 4(a) summarizes the induced modes by bottleneck dimension, where each mode is supported by at least two atoms in the atom graph. For Qwen3-32B, 80 of 94 failure atoms are covered by 11 modes, with Tool Use accounting for the largest share (36.4%). For Qwen3-235B, 28 of 41 failure atoms are covered by 9 modes, concentrated in Tool Use and Evidence Grounding & Output Construction (44.4% each). This pattern is consistent with the nature of AppWorld: tool use is a common bottleneck, while the stronger 235B model produces fewer recurring modes in Task Understanding and Strategy Planning. Its remaining failures are mostly in Evidence Grounding & Output Construction, which often depends on fine-grained evidence handling and output construction.

Figure 4(b) evaluates the runtime usefulness of the most frequently activated modes. Net Improvement Rate measures the fraction of activations that improve performance after subtracting the fraction that hurt performance. For Qwen3-235B, all top-5 activated modes are positive, suggesting more reliable use of failure-mode corrections by stronger agents. For Qwen3-32B, three of the top-5 modes are positive, while two tool-related modes are slightly negative. These results indicate that failure modes can support effective runtime correction, but their benefit still depends on detection precision and whether the corrective signal is activated in the right context. Beyond a frozen memory, we additionally conduct an exploratory self-evolution experiment in which experience is accumulated and updated from test-time trajectories; the relative gain over the no-evolution baseline grows as more tasks are processed (Appendix H).

5.4 Experience Relevance and Aspect Diversity Analysis

We also examine whether our key problem—experience may repeatedly emphasize the same aspect while leaving other aspects under-covered—is alleviated. Since complete ground-truth bottleneck sets are difficult to obtain, we use GPT-5.5 to annotate two proxy metrics on AppWorld. Experience Relevance Rate measures the proportion of injected experiences whose relevance score is at least 3 on a 1–5 scale. Aspect Diversity Ratio is the number of distinct aspects covered by relevant experiences, normalized by all injected experiences.

Figure 5 shows that our method achieves both higher experience relevance and higher aspect diversity than G-Memory and ReMe across the two base models. Skill-Pro also attains strong relevance, but its aspect diversity remains lower than ours. This suggests that organizing experience by bottleneck dimensions helps reduce redundancy and cover more diverse aspects, while stage-aware runtime use

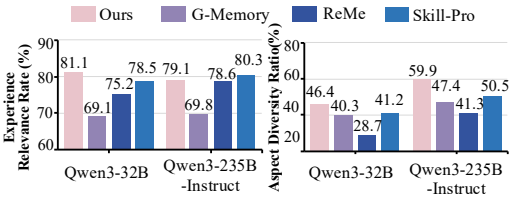


Figure 5: Analysis of experience relevance and aspect diversity on AppWorld.

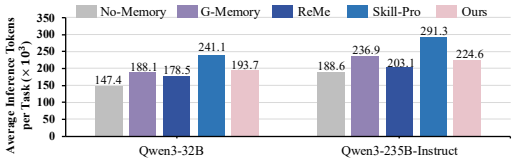


Figure 6: Average total inference tokens per AppWorld task.

improves the relevance of the experience injected into the current execution. On average, our method injects fewer experiences per task than fixed-count baselines (Appendix F).

5.5 Ablation Study

We conduct an ablation study on AppWorld (Table 2) to answer four questions: (1) *Are success skills necessary?* Removing success skills lowers Avg@3 by 3.09 on average across base models, showing the contribution of task-start guidance. (2) *Are failure modes necessary?* Removing failure modes lowers Avg@3 for both models, most notably by 5.44 with Qwen3-235B, showing that runtime correction remains useful even for stronger agents. (3) *Is runtime detection necessary?* Injecting all failure-mode experience from the start lowers Avg@3 for both models, supporting stage-aware activation over static injection. (4) *Is dimension-aware selection necessary?* Mixing all success skills across dimensions and selecting a global top-5 lowers Avg@3 for both models, indicating that bottleneck-aware organization helps retrieve more multifaceted guidance. Appendix C reports the complete ablation results, and Appendix E a dimension-level ablation.

Table 2: Ablation study of our method.

	Variants	Avg@3	Pass@3	Score@3
Qwen3-32B	Full Method	**36.70**	**49.40**	**63.25**
	w/o Success Skills	32.90	46.42	62.16
	w/o Failure Modes	35.11	47.02	62.69
	w/o RT Detection	33.13	46.70	60.49
	w/o Dimension	33.56	46.09	62.51
Qwen3-235B	Full Method	**58.13**	**69.64**	**82.61**
	w/o Success Skills	55.75	68.45	81.60
	w/o Failure Modes	52.69	65.26	79.23
	w/o RT Detection	55.15	67.26	80.68
	w/o Dimension	56.70	67.68	81.13

5.6 Inference Token Cost

We measure inference cost by summing the input and output tokens of all LLM calls for each AppWorld task, counting the complete input context submitted with every request, and averaging over 168 test tasks. As shown in Figure 6, despite the additional runtime detection and intervention, our method maintains a token cost comparable to other experience-memory methods. Timely intervention can prevent the agent from continuing along an erroneous trajectory, reducing unnecessary token consumption and offsetting part of the additional control cost. Skill-Pro consumes more tokens than other methods. The reason is that its temporally extended skills require additional control interactions and may introduce procedural steps that do not lie on the shortest execution path for the current task (Mi et al., 2026). This result further highlights the importance of regulating when and how experience intervenes during execution. We further analyze the sensitivity to the runtime detection interval in Appendix G and provide case studies of the learned experiences in Appendix I.

6 Conclusion

We studied experience reuse in complex agent tasks with heterogeneous, sequentially dependent bottlenecks that are progressively revealed during execution. We introduced an experience harness that organizes trajectories into bottleneck-aware atoms and transforms successful and failed experience into complementary mechanisms for multifaceted task-start guidance and targeted runtime correction. Experiments on AppWorld and BFCL-V3 show superiority over no-memory and prior memory baselines. Future work will explore automatically discovering and adapting bottleneck dimensions, as well as extending experience harness to longer-horizon tasks and evolving environments.

AI USE STATEMENT

In this work, we used generative AI tools to support qualitative data analysis: GPT-5.5 was used as an automatic annotator for the proxy metrics (experience relevance and aspect diversity) reported in Section 5.4. We have not used generative AI tools for any other tasks with required disclosure, including generating synthetic datasets, developing theoretical models or conceptual frameworks, formulating mathematical claims, assisting in proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, translation assistance, cleaning and reformatting datasets, and interpreting results. Additionally, we used generative AI tools for retrieval and discovery, such as finding related work, for assisting with code implementation, and for aiding and polishing the manuscript writing, which fall under tasks with recommended disclosure. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies experience memory for LLM agents and is evaluated on the public AppWorld and BFCL-V3 benchmarks. It does not involve human subjects, new dataset releases, or personally identifiable information. All experiments are conducted in sandboxed benchmark environments. We do not foresee direct harmful applications of the proposed method beyond the general risks associated with LLM agents, such as erroneous actions in interactive environments, which the studied benchmarks are designed to contain.

REPRODUCIBILITY STATEMENT

We describe the full experience construction and deployment pipeline in Section 4 and Algorithm 1, with implementation details, including retrieval settings, merge thresholds, and selection rules, provided in Appendix A.3. Dataset descriptions, train/test splits, and evaluation protocols for AppWorld and BFCL-V3 are provided in Section 5 and Appendix A.1, while descriptions of the compared baselines are provided in Appendix A.2. Complete main results and ablation results across all base models are reported in Appendices B and C.

REFERENCES

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *CoRR*, abs/2512.10696, 2025. doi: 10.48550/ARXIV.2512.10696. URL https://doi.org/10.48550/arXiv.2512.10696.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *ICLR*, 2025.

Shengda Fan, Xin Cong, Zhong Zhang, Yuepeng Fu, Yesai Wu, Hao Wang, Xinyu Zhang, Enrui Hu, and Yankai Lin. Generalizing experience for language agents with hierarchical metaflows. *Advances in Neural Information Processing Systems*, 38:64103–64132, 2026.

GLM. GLM-4.5: agentic, reasoning, and coding (ARC) foundation models. *CoRR*, abs/2508.06471, 2025. doi: 10.48550/ARXIV.2508.06471. URL https://doi.org/10.48550/arXiv.2508.06471.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. In *Forty-third International Conference on Machine Learning*, 2026.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560, 2023. doi: 10.48550/ARXIV.2310.08560. URL https://doi.org/10.48550/arXiv.2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In Sean Follmer, Jeff Han, Jürgen Steimle, and Nathalie Henry Riche (eds.), *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, pp. 2:1–2:22. ACM, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025.

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. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=1i6ZCvflQJ.

Kimi Team. Kimi K2: open agentic intelligence. *CoRR*, abs/2507.20534, 2025. doi: 10.48550/ARXIV.2507.20534. URL https://doi.org/10.48550/arXiv.2507.20534.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.850. URL https://doi.org/10.18653/v1/2024.acl-long.850.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. Skillx: Automatically constructing skill knowledge bases for agents. *CoRR*, abs/2604.04804, 2026. doi: 10.48550/ARXIV.2604.04804. URL https://doi.org/10.48550/arXiv.2604.04804.

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. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025.

Daniel S. Weld. Donald a. norman, the psychology of everyday things. *Artif. Intell.*, 41(1): 111–114, 1989. doi: 10.1016/0004-3702(89)90083-0. URL https://doi.org/10.1016/0004-3702(89)90083-0.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. LLM agents making agent tools. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 26092–26130. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1266/.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: agentic memory for LLM agents. *CoRR*, abs/2502.12110, 2025. doi: 10.48550/ARXIV.2502.12110. URL https://doi.org/10.48550/arXiv.2502.12110.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. *CoRR*, abs/2506.07398, 2025. doi: 10.48550/ARXIV.2506.07398. URL https://doi.org/10.48550/arXiv.2506.07398.

Wenyuan Zhang, Xinghua Zhang, Haiyang Yu, Shuaiyi Nie, Bingli Wu, Juwei Yue, Tingwen Liu, and Yongbin Li. Expseek: Self-triggered experience seeking for web agents. *arXiv preprint arXiv:2601.08605*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024,*

Vancouver, Canada, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 19724–19731. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29946. URL https://doi.org/10.1609/aaai.v38i17.29946.

A Experiment Details

A.1 Dataset Description

AppWorld. AppWorld (Trivedi et al., 2024) is designed to evaluate interactive agents in realistic digital environments. Each task requires an agent to understand a user instruction, interact with multiple application APIs, inspect intermediate execution results, and complete the task by modifying the underlying environment state. Compared with single-step tool-use benchmarks, AppWorld places more emphasis on long-horizon planning, iterative execution, and the ability to avoid unintended side effects. This makes it suitable for evaluating whether an agent can acquire and reuse procedural experience in complex tool-use scenarios.

In our experiments, we use the 90 examples from the AppWorld training split for experience extraction. After extracting experience from these examples, we freeze the experience memory and do not further update it during evaluation. We then evaluate the resulting agent on the 168 examples from the AppWorld test_normal split. This setting allows us to examine whether the extracted experience can generalize from training tasks to unseen test tasks in the same benchmark.

BFCL-v3. The Berkeley Function Calling Leaderboard v3 (BFCL-v3) (Patil et al., 2025) extends function-calling evaluation to multi-turn and multi-step settings. In this benchmark, a model must maintain dialogue context, decide when to call functions, correctly fill function arguments, and incorporate function execution results into subsequent turns. The Base Multi-Turn subset contains foundational multi-turn tool-use scenarios in which the required information can be obtained from the conversation history or previous tool outputs. It provides a controlled setting for evaluating multi-turn function-calling competence.

For BFCL-v3, we use the Base Multi-Turn category. We randomly sample 180 examples from this category and split them into 60 examples for experience extraction and 120 examples for testing. The two subsets are non-overlapping. After experience is extracted from the 60 training examples, it is frozen before evaluation on the 120 held-out test examples. This protocol is used to assess whether our method can transfer reusable tool-use experience to unseen multi-turn function-calling tasks.

A.2 Baseline Description

AWM (Agent Workflow Memory) (Wang et al., 2025) focuses on inducing reusable workflows from past agent trajectories. Rather than storing complete trajectories as examples, AWM abstracts common sub-routines from previous experiences, such as repeated operation patterns or task-specific procedures. These workflows are written in natural language and omit instance-specific details, making them more reusable across related tasks. At inference time, the induced workflows are provided to the agent as auxiliary procedural memory to guide future action generation. AWM can be applied in an offline setting, where workflows are induced from training examples before evaluation, or in an online setting, where workflows are incrementally induced from successful test-time experiences.

ExpeL (Zhao et al., 2024) is an experiential learning method for LLM agents. Instead of updating model parameters, ExpeL enables an agent to learn from its own interaction history. During training, the agent collects task-solving trajectories through trial and error, including both successful and failed attempts. It then extracts high-level natural language insights from these experiences, such as common failure patterns and useful strategies. At inference time, ExpeL augments the agent prompt

Table 3: Complete main results on AppWorld and BFCL-V3 across four base models.

Model	Method	AppWorld			BFCL-V3		Cross-Benchmark	
		Avg@3	Pass@3	Score@3	Avg@3	Pass@3	Avg@3	Pass@3
Qwen3-32B	No Memory	25.59	41.66	60.59	42.78	48.33	34.19	44.99
	ExpeL	30.01	46.42	60.30	45.27	45.83	37.64	46.12
	AWM	26.98	45.83	59.96	45.83	49.17	36.40	47.50
	G-Memory	32.57	47.61	62.86	46.79	50.83	39.68	49.22
	ReMe	31.61	47.02	61.86	45.54	49.17	38.57	48.10
	MetaFlowLLM	29.96	45.23	59.64	45.27	49.17	37.61	47.20
	Skill-Pro	30.16	45.83	60.56	46.66	52.50	38.41	49.16
	Ours	**36.70**	**49.40**	**63.25**	**49.44**	**52.50**	**43.07**	**50.95**
Qwen3-235B-A22B-Instruct	No Memory	49.40	66.66	76.73	49.04	60.83	49.22	63.75
	ExpeL	52.10	67.85	79.91	55.55	63.33	53.83	65.59
	AWM	51.16	68.45	78.54	51.67	61.67	51.41	65.06
	G-Memory	55.13	69.64	80.54	53.89	64.16	54.51	66.90
	ReMe	54.06	67.26	78.65	53.01	62.50	53.53	64.88
	MetaFlowLLM	55.35	68.45	80.35	53.61	63.33	54.48	65.89
	Skill-Pro	56.15	**70.23**	81.90	54.44	64.16	55.29	67.19
	Ours	**58.13**	69.64	**82.61**	**57.50**	**65.83**	**57.82**	**67.74**
GLM-4.6	No Memory	60.70	70.23	81.42	74.83	78.33	67.76	74.28
	ExpeL	63.21	72.02	83.20	76.19	80.83	69.70	76.42
	AWM	61.12	70.83	82.10	75.27	79.16	68.19	74.99
	G-Memory	64.01	73.80	84.20	77.03	80.00	70.52	76.90
	ReMe	64.73	71.42	83.20	78.11	80.00	71.42	75.71
	MetaFlowLLM	65.07	73.21	84.80	75.83	79.16	70.45	76.18
	Skill-Pro	63.09	71.42	82.18	78.61	80.83	70.85	76.12
	Ours	**67.65**	**75.59**	**85.95**	**79.72**	**81.67**	**73.69**	**78.63**
Kimi-K2-Instruct	No Memory	49.60	67.26	77.53	60.55	68.33	55.07	67.79
	ExpeL	51.79	70.24	79.42	61.11	70.00	56.45	70.12
	AWM	50.79	68.45	78.95	60.83	69.17	55.81	68.81
	G-Memory	53.37	70.83	81.56	61.67	72.50	57.52	71.66
	ReMe	56.55	69.64	80.17	62.22	70.83	59.38	70.23
	MetaFlowLLM	57.53	73.21	82.32	62.50	73.33	60.01	73.27
	Skill-Pro	57.93	70.83	80.95	62.22	**76.66**	60.07	73.74
	Ours	**62.89**	**75.00**	**83.90**	**64.44**	73.33	**63.66**	**74.16**

with the extracted insights and retrieves the most similar successful trajectory from the experience pool as in-context examples. In this way, ExpeL combines abstract procedural knowledge with concrete demonstrations from past tasks.

GMemory (Zhang et al., 2025) is a hierarchical memory method originally designed for LLM-based multi-agent systems. It organizes past collaborative experiences into a structured graph-based memory hierarchy, including high-level insights, task-level query information, and fine-grained interaction trajectories. Given a new task, GMemory retrieves both generalizable knowledge and relevant historical interaction patterns through memory traversal, allowing the agent system to reuse prior experience for planning and coordination. Specifically, for generalizable insights, we inject the top five entries with the highest similarity scores into the context, while for interaction trajectories, we inject the top-ranked entry. After task execution, the memory hierarchy can be updated with newly collected trajectories, supporting continual evolution of the agent memory. For comparison, GMemory represents a structured graph-based memory baseline.

ReMe (Cao et al., 2025) is a dynamic procedural memory method for experience-driven agent evolution. It addresses the limitation of treating memory as a static append-only archive by introducing a complete memory lifecycle. First, ReMe distills fine-grained procedural experiences from trajectories by identifying success patterns, analyzing failure causes, and generating comparative insights. Second, it retrieves and adapts relevant experiences for a new task through context-adaptive reuse, including scenario-aware indexing and rewriting; in our implementation, we inject the top five most relevant retrieved experiences into the context. Third, it refines the memory pool over time by adding useful new experiences and pruning low-utility or outdated memories. Therefore, ReMe serves as a strong dynamic memory baseline that explicitly models experience acquisition, reuse, and refinement.

MetaFlowLLM (Fan et al., 2026) is a hierarchical procedural experience method that generalizes successful agent trajectories into reusable executable workflows. It represents each workflow, termed a MetaFlow, as a sequence of static steps that directly execute deterministic operations and dynamic steps that delegate task-dependent subtasks to the underlying agent. During experience construction, MetaFlowLLM converts successful trajectories into leaf-level workflows, recursively merges similar workflows, and validates the resulting abstractions, thereby organizing them into a hierarchy ranging from task-specific trajectories to more general procedures. At runtime, it traverses this hierarchy to select the deepest MetaFlow applicable to the current task and injects the selected workflow into the agent's execution process. Rather than providing all experience only at the beginning, the workflow intervenes throughout execution: static steps directly determine reusable tool operations, while dynamic instructions are injected at their corresponding stages to guide the agent in completing uncertain subtasks and producing variables required by subsequent steps. Therefore, MetaFlowLLM serves as a strong runtime injection baseline that combines hierarchical experience abstraction with process-level procedural injection.

Skill-Pro (Mi et al., 2026) is a non-parametric procedural skill learning method for experience-driven LLM agents. It addresses the limitation of storing past trajectories or reflections as passive memories by converting interaction experience into reusable executable skills that can be injected into the agent's decision-making process. Each skill is represented with an activation condition, an execution procedure, and a termination condition, specifying when the skill should be used, what procedural guidance it provides, and when it should stop intervening. During training, Skill-Pro refines the skill pool from collected trajectories through semantic gradients, PPO-style skill verification, and score-based pruning. During inference, the agent matches the current state against the activation conditions of stored skills, selects the most relevant skill, and injects its execution procedure into the LLM context to guide the next action generation. The selected skill continues to condition the agent's decisions until its termination condition is satisfied, after which control returns to the general policy or another applicable skill. Therefore, Skill-Pro serves as a strong procedural skill learning baseline that explicitly models experience abstraction, runtime skill injection, skill-conditioned decision making, and long-term skill pool maintenance without updating the parameters of the underlying LLM.

A.3 Implementation Details

Trajectory-to-Atom Extraction. We use the official evaluator of each benchmark to determine whether a trajectory succeeds. For AppWorld, an evaluator runs database-state-based unit tests over the final environment state, checking whether the requested task goal is satisfied and whether the agent avoids unintended state changes. For BFCL-v3 Multi Turn, an evaluator executes the model-generated function calls and the ground-truth function calls on the backend system and performs checks after each turn. A trajectory must pass both state-based checks, which compare the public backend states against the ground-truth states, and response-based checks, which verify that the required execution results are covered by the model's execution path.

For a failed attempt at round r, we extract a candidate failure atom α_r^- with a repair hint e_r^- in it. The next attempt is then executed with e_r^- injected into the agent context. We allow at most 5 such repair-based retries for each training task. After observing the next trajectory τ_{r+1}, an LLM judge determines whether the repair experience contributes to the new attempt. We denote this binary judgment as γ_r. A candidate failure atom is retained for experience induction only when $\gamma_r = 1$, which indicates that the injected experience made the agent materially closer to completing the original task goal.

Atom-to-Experience Induction. Canonicalization is an LLM-based, dimension-specific rewriting step. Its objective is to reduce dependence on an individual task instance while retaining the conditions under which the experience remains useful. The prompts require removing incidental instance details, including names, contact information, record identifiers, exact dates, literal object names, and nonessential numerical values. They additionally require rewriting instance descriptions into reusable reasoning or operational patterns, rather than merely replacing named entities with placeholders. The retained structure is dimension-specific: goal types and constraints for task understanding, ordered steps for workflow control, tool names and contracts for tool use, and evidence alignment and output fidelity for grounding. Domain- or tool-specific details are retained when they define the pattern's applicability. Canonicalization should therefore be understood as a procedure for reducing instance

dependence while preserving actionable structure, rather than a guarantee that all task-specific information has been eliminated.

We implement atom clustering with local embedding-based retrieval followed by LLM-based merge judgment. Specifically, each canonicalized atom is encoded with `Qwen3-Embedding-8B`. For each atom, we compute cosine similarity in the same atom type and bottleneck dimension, retrieve the top-7 nearest atoms, and combine them with the current atom to form a neighborhood of 8 atoms. This local comparison strategy avoids exhaustive all-pair comparisons while allowing repeated neighborhood overlaps to accumulate evidence for reliable merging.

The merge judgments are accumulated into a graph. For each atom pair (α_i, α_j), c_{ij} denotes the number of neighborhoods in which the pair co-occurs, and m_{ij} denotes the number of times the pair is judged mergeable. We use $s_{ij} = m_{ij}/c_{ij}$ as the merge score and add an edge when $c_{ij} \geq 3$ and $s_{ij} \geq 0.6$.

Stage-Aware Experience Use. For task-start success guidance, we implement $\mathrm{Retrieve}(x, M^+)$ with embedding-based task retrieval. Given a test task x, we encode its instruction using `Qwen3-Embedding-8B`. We then compute cosine similarity between this embedding and the instruction embeddings of training tasks associated with induced success skills. The top-3 most similar training tasks are retrieved, and their success skills are used as the candidate pool. We instantiate $\mathrm{Select}_{\mathrm{LLM}}$ as a dimension-aware skill selection step. The LLM selector receives the test instruction and the retrieved candidate skills, and selects useful skills separately for the four bottleneck dimensions. For the tool-use dimension, the selector is allowed to select any number of skills, including zero. For each of the remaining dimensions, the selector can select at most one skill, and may select none when no candidate is relevant. The final selected set S_x is injected before the first action and kept unchanged during the execution.

B Full Main Results

The main paper reports results for three base models for compactness. Table 3 presents the complete results across all four base models, additionally including Kimi-K2-Instruct (Team, 2025) beyond the three models reported in the main text. The full results confirm the same overall trend: our method achieves the highest Avg@3 on both AppWorld and BFCL-V3 for every base model. Compared with the corresponding no-memory baseline, it yields relative Cross-Benchmark Avg@3 improvements of 25.97%, 17.47%, 8.75%, and 15.60% for Qwen3-32B, Qwen3-235B-A22B-Instruct, GLM-4.6, and Kimi-K2-Instruct, respectively. On Kimi-K2-Instruct, our method improves the Cross-Benchmark Avg@3 from 55.07 to 63.66 and also achieves the highest Cross-Benchmark Pass@3 of 74.16. Although Skill-Pro obtains a higher Pass@3 on BFCL-V3, our method performs substantially better in Avg@3 on both benchmarks, indicating more consistent improvements across repeated executions. Overall, the additional Kimi-K2-Instruct results further demonstrate that the benefits of our bottleneck-aware experience construction and runtime intervention generalize across different model families and scales.

C Full Ablation Results

The main paper reports component-level ablations for the two Qwen base models for compactness. Table 4 presents the complete results, additionally including GLM-4.6.

For GLM-4.6, removing success skills, failure modes, runtime detection, or dimension-aware selection consistently lowers Avg@3. Removing success skills produces the largest decrease, while all four variants underperform the full method, extending the component-level trends observed with the two Qwen models.

D Definitions of Four Dimensions

Task Understanding. This dimension captures how the agent frames the task objective, success criterion, critical constraints, target relations, and task scope. It concerns whether the agent correctly understands what goal should be achieved, what conditions determine success or failure, and which task-specific constraints or interpretations must be preserved.

Table 4: Complete component-level ablation results on AppWorld across three base models.

	Variant	Avg@3	Pass@3	Score@3
Qwen3-32B	Full Method	**36.70**	**49.40**	**63.25**
	w/o Success Skills	32.90	46.42	62.16
	w/o Failure Modes	35.11	47.02	62.69
	w/o RT Detection	33.13	46.70	60.49
	w/o Dimension	33.56	46.09	62.51
Qwen3-235B	Full Method	**58.13**	**69.64**	**82.61**
	w/o Success Skills	55.75	68.45	81.60
	w/o Failure Modes	52.69	65.26	79.23
	w/o RT Detection	55.15	67.26	80.68
	w/o Dimension	56.70	67.68	81.13
GLM-4.6	Full Method	**67.65**	**75.59**	**85.95**
	w/o Success Skills	63.49	72.62	84.91
	w/o Failure Modes	64.08	73.21	85.45
	w/o RT Detection	65.07	72.80	85.01
	w/o Dimension	64.28	73.40	84.74

Table 5: Dimension-level ablation on AppWorld. Each variant removes all test-time success skills and failure modes from one bottleneck dimension.

Variant	Avg@3	Pass@3	Score@3
Qwen3-32B			
Full Method	**36.70**	**49.40**	63.25
w/o Task Understanding	33.13	46.43	62.12
w/o Tool Use	35.80	49.38	**63.75**
w/o Evidence Grounding	33.33	45.23	62.08
w/o Strategy Planning	33.54	45.06	61.50
Qwen3-235B-A22B-Instruct			
Full Method	**58.13**	**69.64**	**82.61**
w/o Task Understanding	57.14	66.67	82.08
w/o Tool Use	57.34	67.85	82.38
w/o Evidence Grounding	55.48	66.46	81.17
w/o Strategy Planning	57.43	69.08	82.28

Strategy Planning. This dimension captures the overall execution route for solving the task, including workflow decomposition, dependency ordering, verification timing, and search or pagination strategy. It concerns whether the agent organizes the task into an appropriate sequence of steps and gathers necessary evidence before taking irreversible or goal-completing actions.

Tool Use. This dimension captures concrete tool, API, or application operation usage, including which tool family or function is needed, what operational role it serves, how it should be invoked step by step, and which parameters, credentials, schemas, or preconditions must be satisfied before the call. It concerns whether the agent selects and uses tools in a schema-aware and context-appropriate manner.

Evidence Grounding & Output Construction. This dimension captures how the agent locates, verifies, and interprets the target object or evidence, and how it transforms that grounded evidence into the required final answer, action, or action payload. It concerns whether the agent bases its final behavior on the correct observed information rather than on missing, stale, hallucinated, or misinterpreted evidence.

E Dimension-Level Ablation

We conduct an additional dimension-level ablation on AppWorld using the two Qwen base models. For each variant, we mask all success skills and failure modes associated with one bottleneck dimension during test-time use. Table 5 shows that removing any dimension lowers Avg@3 for both

Table 6: Average number of injected experiences per task. Our method uses dynamic selection, while fixed-count baselines inject five experiences per task.

Base Agent	AppWorld	BFCL-V3
Qwen3-32B	7.2	2.7
Qwen3-235B-A22B-Instruct	4.6	2.4
GLM-4.6	3.8	3.5
Fixed-count baseline	5.0	5.0

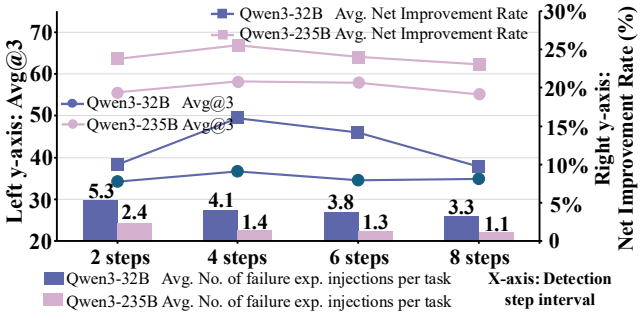


Figure 7: Sensitivity to runtime failure-mode detection intervals on AppWorld.

models, suggesting that the four dimensions capture complementary experience. For Qwen3-32B, removing Task Understanding, Evidence Grounding, or Strategy Planning leads to larger drops than removing Tool Use, indicating that the smaller model benefits strongly from high-level task framing and execution organization. For Qwen3-235B, the largest drop comes from removing Evidence Grounding, which is consistent with our failure-mode analysis: stronger models make fewer recurring planning errors, but still rely on fine-grained evidence and output construction.

F Injected Experience Statistics

We further report the average number of experiences injected by our method at test time. For baselines that require a fixed number of injected experiences, we use five experiences per task. The results are shown in Table 6. Since our method lets the agent omit retrieved experiences that are unsuitable for the current task, the average number of injected experiences varies across models and datasets and is often lower than the fixed-count setting.

G Sensitivity to Runtime Detection Interval

We study how often the agent should run failure-mode detection during execution. Smaller intervals detect deviations earlier but risk unnecessary interventions, while larger intervals reduce interventions but may delay correction. Figure 7 shows this trade-off on AppWorld. For both Qwen3-32B and Qwen3-235B, an interval of 4 steps achieves the best or near-best Avg@3 and net improvement rate. In contrast, detecting every 2 steps leads to more frequent injections without the best task performance, while intervals of 6 or 8 steps reduce intervention frequency but also weaken correction. These results suggest that stage-aware correction benefits from moderate rather than maximal detection frequency.

H Exploratory Self-Evolution Experiment

We additionally conduct a preliminary self-evolution experiment to examine whether the proposed experience harness method can improve as more test-time trajectories become available. This experiment is intended as an exploratory analysis of the paper. We therefore do not position it as a full comparison against existing self-improvement or continual-learning methods. In this setting, the agent starts with an empty experience pool and processes the 168 AppWorld test tasks sequentially without

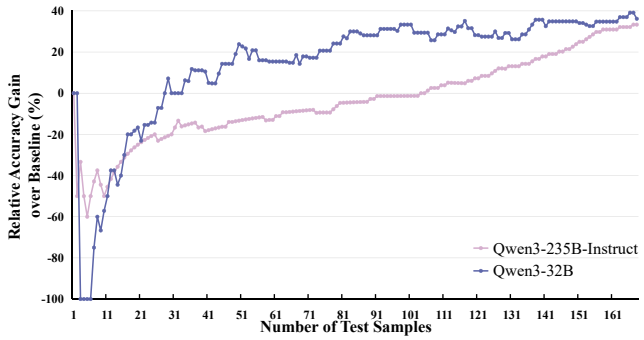


Figure 8: Relative accuracy gain over the no-evolution baseline as the number of processed test samples increases. The curves show Qwen3-32B and Qwen3-235B-Instruct under the same self-evolution protocol, where newly collected trajectories are periodically converted into reusable experiences for later tasks.

using any training-set trajectories. After every batch of 16 tasks, successful and failed trajectories from the completed tasks are summarized into success and failure atoms, which are then aggregated into experience-level guidance following the same graph-based procedure used in our main method. The updated experiences are used for subsequent batches. We report the evolving performance trend to illustrate the potential of the experience harness to support test-time accumulation and reuse of task-solving knowledge.

We measure the self-evolution trend using the relative accuracy gain over the corresponding no-evolution baseline. At each test position, we compute the cumulative accuracy of the self-evolving agent over all tasks processed so far, and compare it with the cumulative accuracy of the baseline over the same ordered task prefix. The reported value is calculated as $(\mathrm{Acc}_{\mathrm{self}} - \mathrm{Acc}_{\mathrm{base}})/\mathrm{Acc}_{\mathrm{base}}$, expressed as a percentage. As shown in Figure 8, both Qwen3-32B and Qwen3-235B-Instruct exhibit an increasing relative gain as more test samples are processed, suggesting that the accumulated experiences become progressively more useful for subsequent tasks.

I Case Study

We present a case study on AppWorld to illustrate the learned experience memory and its stage-aware use during execution.

Figure 9 illustrates a representative AppWorld task that exposes multiple dependent bottlenecks. The agent must first establish valid credentials and access tokens, then ground the file-system operation through the correct API to read `old_songs.txt`, and finally use the extracted evidence to remove matching songs from all Spotify playlists and create the target archive playlist. Later bottlenecks depend on earlier ones: without authenticated file access, the agent cannot obtain the song evidence; without grounding that evidence to Spotify song identifiers, it cannot complete the playlist modification correctly.

The comparison shows that ReMe retrieves several experiences that are mostly concentrated on Spotify metadata or metric-analysis scenarios, such as genre filtering, play-count retrieval, recommendation analysis, and like-count lookup. Although one retrieved experience is relevant to API/schema validation, the retrieved set leaves important bottlenecks uncovered, especially file-system access and cross-playlist archive construction. As a result, ReMe eventually finds a runnable path but only partially completes the task.

In contrast, our method provides experiences that cover different roles in the task. The selected success skills guide cross-playlist task understanding, end-to-end workflow planning, credential setup, and grounding file evidence into Spotify playlist actions. During execution, failure modes further provide targeted corrections for missing authentication and incorrect API parameter names. This case demonstrates how multifaceted guidance and stage-aware correction help the agent handle complex tasks whose bottlenecks appear at different stages of execution.

Beyond this complete task example, we provide representative failure modes for both Qwen3-32B and Qwen3-235B, together with one success-skill example from each bottleneck dimension for both models. The failure-mode examples are shown in Figures 10–19, and the success-skill examples are shown in Figures 20–27.

Task Instruction:
Go through all my Spotify playlists and remove all the songs from them that are in "~/documents/personal/old_songs.txt" from my file system and put them in a new playlist named "Archived Songs".

Bottleneck of this task:
📌Bottleneck 1: Establish Required Credentials and Access Tokens
📌Bottleneck 2: Ground File Access Through the Correct File-System API
📌Bottleneck 3: Complete Cross-Playlist Song Removal and Archive Creation

Ours:

Success Skill 1: Relevant: Cross-playlist task understanding For Bottleneck 3 📌
Core_principle:
Understand the goal as applying the same filtering rule across all relevant collections, rather than treating each collection separately.
Critical_constraints: The filtering criterion must be applied uniformly; no relevant collection should be omitted.
Misread_to_avoid: Treating the task as modifying only one playlist or handling playlists in isolation.

Success Skill 2: Relevant: Workflow guiding the model from data retrieval to playlist updates
Workflow_steps:
Step1: Check available APIs and access requirements;
Step2: Authenticate first;
Step3: Retrieve complete data with pagination;
Step4: Isolate target items by reliable criteria;
Step5: Apply conditional updates safely.
Critical_checkpoints:
Ensure complete data retrieval and consistent filtering before modifying collections.

Success Skill 3: Relevant: Authentication and access-token setup For Bottleneck 1 📌
Tool_usage_steps:
Step1: Retrieve valid credentials;
Step2: Call the login API with the required fields;
Step3: Extract and reuse the returned access token.
Critical_preconditions: Credentials must be available and correctly formatted; the login response must provide an access token.

Success Skill 4: Relevant: Grounding old_songs.txt evidence to Spotify playlist actions For Bottleneck 2 📌
Grounding_principle: Ground target songs by reading the file evidence, resolving each song to exact Spotify identifiers, and matching them against playlist contents.
Output_alignment_rule: Add the matched songs to the archive playlist and remove them from every original playlist where they appear.
Failure_to_avoid: Do not rely on bulk assumptions; verify item identity and preserve non-matching songs.

Injected Failure Mode_006 at step 2: Relevant: Runtime correction for unauthentica ted file access For Bottleneck 1 📌
Mode Name: Missing Authentication
Experience:
When accessing a protected app or resource for the first time in a task, treat a uthentication as an explicit prerequisite rather than an implicit default. If the operation depends on a session, token, or credential, obtain and pass it first be fore probing the protected endpoint …

Injected Failure Mode_004 at step 9: Relevant: Runtime correction for file_system parameter mismatch
Mode Name: Wrong Parameter Name.
Experience:
If an API call fails at validation or the parameter names are not fully certain, verify the exact argument names for that specific endpoint instead of reusing names from a similar API by analogy …

ReMe:

Experience 1: Irrelevant: Spotify genre/year filtering related
When to use: When filtering Spotify songs by genre and release year.
Content: Use `search_songs` instead of `show_recommendations` after verifying that its response provides fields such as `genre` and `release_date`; apply precise query parameters and handle pagination for complete retrieval.

Experience 2: Relevant: API/schema validation for playlist operations in Spotify For Bottleneck 1 📌
When to use: When retrieving Spotify song/playlist metadata or metrics.
Content: Check API availability and parameters with `show_api_docs`; if a metric is not directly exposed, use related data such as playlist song IDs with the appropriate metadata endpoint.

Experience 3: Irrelevant: Spotify play count retrieval related
When to use: When retrieving user-specific Spotify play counts.
Content: Verify which endpoint exposes play-count data; do not infer play frequency from indirect metadata such as reviews or likes, and use the most direct metric endpoint available.

Experience 4: Irrelevant: Recommendations and artist frequency related
When to use: When analyzing personalized Spotify recommendations.
Content: Use `show_recommendations` with pagination to collect all results, aggregate artist metadata across recommended songs, and count artist frequency to identify dominant recommendation patterns.

Experience 5: Irrelevant: Spotify like-count metric retrieval related
When to use: When finding the most-liked song in Spotify playlists.
Content: Use direct metrics such as `like_count` from `show_song`; avoid inferring popularity from indirect signals like playlist frequency.

Figure 9: Case study on a multi-stage AppWorld task. The task requires the agent to access a file, ground its content to Spotify songs, and complete cross-playlist removal and archive creation.

Failure Mode Example: Wrong Resource ID Mapping

locus_bucket:
Evidence Grounding & Output Construction

concept_name:
Wrong Resource ID Mapping

working_definition:
A failure mode where systems or agents use incorrect, unverified, or misinterpreted identifiers to access or manipulate resources, resulting in access failures or incorrect operations.

shared_discrepancy_kernel:
A mismatch between expected and actual resource identifiers due to overreliance on assumptions, incomplete or incorrect data sources, or misunderstanding of identifier-to-resource mapping.

shared_mechanism:
The agent uses a resource identifier directly without validation or context-aware verification, leading to failed operations such as login failure, file access errors, or message send failures due to non-existent or mismatched identifiers.

experience:
If the next action depends on a resource identifier, consider checking whether the identifier has been resolved to the exact form the target API expects. Names, guessed paths, hardcoded contacts, or loosely related credentials are often only partial clues. It is usually safer to validate the identifier against the relevant source, such as a contact list, file listing, account record, or returned object, before acting.

Figure 10: Failure mode example from Qwen3-32B: Wrong Resource ID Mapping

Failure Mode Example: Missing Authentication

locus_bucket:
Tool Use

concept_name:
Missing Authentication

working_definition:
A failure mode where an agent attempts to access protected APIs or services without valid authentication credentials, leading to unauthorized access errors and disrupted task flow.

shared_discrepancy_kernel:
Violation of the authentication principle: invoking protected API functions without verifying or providing the necessary access token or credentials.

shared_mechanism:
The agent executes an API call to a service that requires authentication but does not first obtain or include a valid access token, leading to an unauthorized (e.g., 401) error, which halts the current operation and delays or blocks further progress in the task.

experience:
If a protected API call is about to happen, consider checking whether the correct app-specific authenticated session or access token is already available. After a 401 or unauthorized signal, it may be better to treat authentication as still unresolved and refresh the credential for that same service before continuing with related calls.

Figure 11: Failure mode example from Qwen3-32B: Missing Authentication

Failure Mode Example: Invalid API Call

locus_bucket:
Tool Use

concept_name:
Invalid API Call

working_definition:
A failure mode where the agent invokes an API method that does not exist, is incorrectly configured, or improperly used, leading to execution failure or task disruption.

shared_discrepancy_kernel:
Mismatch between the agent's assumed API capabilities and the actual available or required API structure, including incorrect method names, invalid parameters, or bypassing necessary authentication or validation.

shared_mechanism:
The agent attempts to execute an API call based on assumptions about its availability or structure, but the call fails due to the method not existing or arguments not being valid, causing the task to either stall, fail immediately, or require backtracking to correct the API usage.

experience:
If an API call is based partly on memory or guesswork, consider checking the available API surface before proceeding. It may help to confirm the exact method exists, its arguments match the documented contract, and the return structure supports the intended next step. When the method name, parameter type, or capability is uncertain, inspecting the API surface is often safer than improvising a call.

Figure 12: Failure mode example from Qwen3-32B: Invalid API Call

Failure Mode Example: Text-in-Code Error

locus_bucket:
Tool Use

concept_name:
Text-in-Code Error

working_definition:
A failure mode where natural language or explanatory text is inserted into executable code blocks, violating syntax requirements and causing execution to fail.

shared_discrepancy_kernel:
A mismatch between the intent to explain or debug and the requirement to maintain syntactically correct executable blocks of code.

shared_mechanism:
The inclusion of non-executable text in code contexts triggers syntax errors, halting execution and preventing any intended progress in task fulfillment.

experience:
If the next response is meant to be executable code, consider keeping the code block syntactically clean and containing only executable code. Explanations or planning text may be safer outside the executable block so that the code itself stays runnable.

Figure 13: Failure mode example from Qwen3-32B: Text-in-Code Error

Failure Mode Example: Unverified Fields

locus_bucket:
Evidence Grounding & Output Construction

concept_name:
Unverified Fields

working_definition:
A failure mode where an agent assumes the availability or structure of specific data fields without validating their existence or format, leading to errors or misinterpretations.

shared_discrepancy_kernel:
The agent operates under the assumption that required fields or structures are consistently available and correctly formatted, without performing necessary checks or fallbacks.

shared_mechanism:
This assumption leads to direct data access (e.g., via dictionary keys) that fails when the expected fields are missing or malformed, typically causing crashes (like KeyError), incorrect outputs, or halted execution.

experience:
If a step depends on a specific field, consider checking the actual response schema before relying on that field for filtering, ranking, or answer construction. It may help to confirm the field is present on the current object type and its meaning really matches the task requirement. If the field is missing or only a proxy signal, a validated fallback source may be safer than continuing with direct access.

Figure 14: Failure mode example from Qwen3-32B: Unverified Fields

Failure Mode Example: Missing Authentication

locus_bucket:
Tool Use

concept_name:
Missing Authentication

working_definition:
A failure mode where an agent attempts to access a protected resource without fulfilling authentication requirements, leading to access denial.

shared_discrepancy_kernel:
The agent assumes access to a protected resource is permissible without authentication, violating the system's access control principle that requires verified credentials.

shared_mechanism:
The agent invokes a restricted API or operation without providing valid credentials, resulting in a 401 unauthorized error, which blocks further progress and necessitates recovery through authentication debugging.

experience:
When accessing a protected app or resource for the first time in a task, treat authentication as an explicit prerequisite rather than an implicit default. If the operation depends on a session, token, or credential, obtain and pass it first before probing the protected endpoint. For example, before listing private files or retrieving protected contact data, first obtain the required credential or access token and include it in the API call.

Figure 15: Failure mode example from Qwen3-235B: Missing Authentication

Failure Mode Example: Premature Tool Use

locus_bucket:
Tool Use

concept_name:
Premature Tool Use

working_definition:
A failure mode where an agent uses a tool or API without validating its requirements, capabilities, or dependencies, leading to avoidable errors.

shared_discrepancy_kernel:
The agent acts on assumed tool readiness without verifying preconditions, contradicting the control principle that tool use should be preceded by contract validation.

shared_mechanism:
The agent attempts to invoke a tool or use a dependency without prior validation, triggers a recoverable error, and incurs delays through corrective retries or alternative workarounds.

experience:
Before using a new tool, endpoint, or dependency, check that its method exists, its prerequisites are satisfied, and any required imports, contracts, or login requirements are already in place. A quick validation step is often safer than discovering those assumptions only after the call fails. For example, before calling an unfamiliar API, verify its available methods, required parameters, login requirements, and needed imports instead of assuming they are already valid.

Figure 16: Failure mode example from Qwen3-235B: Premature Tool Use

Failure Mode Example: Wrong Parameter Name

locus_bucket:
Tool Use

concept_name:
Wrong Parameter Name

working_definition:
A failure mode where incorrect parameter names are used in API calls, leading to validation errors.

shared_discrepancy_kernel:
Assumption of parameter naming consistency across APIs or intuitive naming, despite explicit documentation or prior feedback indicating otherwise.

shared_mechanism:
The agent selects parameter names based on guesswork or analogical reasoning, resulting in invalid API requests that fail validation and block further progress.

experience:
If an API call fails at validation or the parameter names are not fully certain, verify the exact argument names for that specific endpoint instead of reusing names from a similar API by analogy. Small naming differences often matter even when the operation looks conceptually the same. For example, verify endpoint-specific parameter names before calling an API, rather than guessing names like receiver_email, path, or source_path by analogy.

Figure 17: Failure mode example from Qwen3-235B: Wrong Parameter Name

Failure Mode Example: Premature ID Grounding

locus_bucket:
Evidence Grounding & Output Construction

concept_name:
Premature ID Grounding

working_definition:
A failure mode where an agent incorrectly resolves identity or existence claims based on incomplete or unverified data sources, leading to erroneous assumptions about entity presence or validity.

shared_discrepancy_kernel:
The agent assumes definitive knowledge of identity or existence from partial evidence, violating the control principle of verifying critical attributes through multiple or authoritative sources before drawing conclusions.

shared_mechanism:
Faced with uncertain or incomplete data, the agent defaults to brittle identity matching or emptiness inference, which propagates as missed opportunities, silent approval failures, or redundant exploration due to unfounded confidence in incomplete grounding.

experience:
When a task depends on identifying a person, object, or relationship, avoid grounding the decision on a single partial clue such as an empty field, a loose name match, or one incomplete source. If the identity is still uncertain, check an additional authoritative source before acting on that assumption.

Figure 18: Failure mode example from Qwen3-235B: Premature ID Grounding

Failure Mode Example: Schema Blindness

locus_bucket:
Evidence Grounding & Output Construction

concept_name:
Schema Blindness

working_definition:
A failure mode where an agent acts on assumed data structures without validating actual schema, leading to incorrect interpretation or processing of evidence.

shared_discrepancy_kernel:
Assumed structural consistency of external data (e.g., field presence, format, type) without runtime validation or defensive checking against actual observed structure.

shared_mechanism:
The agent accesses or parses data based on anticipated schema, but when the actual structure diverges—due to variability in APIs, formats, or metadata—the operation fails silently or throws an error, disrupting downstream logic.

experience:
If the next step depends on fields, nested paths, or timestamp formats from API or file data, first inspect a real sample and confirm the exact structure before writing logic around it. It is safer to validate field presence, type, and format explicitly than to assume the schema is stable across objects or endpoints.

Figure 19: Failure mode example from Qwen3-235B: Schema Blindness

Success Skill Example: Task Understanding

axis_name:
Task Understanding

skill_type:
principle

applicability:
Tasks requiring the identification of the top item based on a metric across nested or grouped data structures.

core_principle:
Frame the task by clearly distinguishing the target of the metric (e.g., individual items) from the structure in which they are organized (e.g., collections or groups).

critical_constraints:
- Ensure the metric is applied to the correct level of granularity (e.g., items, not groups).
- Accurately retrieve and aggregate the metric across all relevant nested or grouped structures.

misread_to_avoid:
Misinterpreting the metric as a property of the container (e.g., collection or group) rather than the individual items within it.

Figure 20: Success skill example from Qwen3-32B: Task Understanding

Success Skill Example: Strategy Planning

axis_name:
Strategy Planning

skill_type:
workflow_steps

applicability:
This skill is applicable to tasks that require systematically identifying, filtering, and acting on a subset of entities or items within a larger collection, while avoiding redundant or invalid operations.

workflow_steps:
- Authenticate and establish access to the system or platform.
- Retrieve the full set of relevant items or entities from the system.
- Filter or identify the subset of items/entities that meet the task's criteria.
- Compare the identified subset with existing states or relationships to avoid redundant actions.
- Perform the required action (e.g., follow, unfollow, update, remove) on the filtered subset, with error handling to skip invalid or already-processed cases.
- Complete the task once all valid actions are executed, ensuring no unnecessary or duplicate operations.

critical_checkpoints:
- Ensure all data is retrieved before performing any modifications to avoid incomplete or inconsistent results.
- Use comparison or set-based logic to identify only the necessary actions and avoid redundant operations.
- Implement error handling to gracefully skip invalid or already-processed items, ensuring robust execution.

Figure 21: Success skill example from Qwen3-32B: Strategy Planning

Success Skill Example: Tool Use

axis_name:
Tool Use

skill_type:
tool_usage_steps

applicability:
This skill applies to tools that perform directory compression and optional cleanup, ensuring correct configuration of source, target, and operation flags.

tool_usage_steps:
- Identify and specify the source directory to be compressed.
- Define the target file path and format for the compressed archive.
- Provide necessary authentication credentials for the file system operation.
- Optionally configure the tool to delete the source directory after compression.
- Optionally enable overwriting of an existing target archive file if needed.

critical_preconditions:
- A valid source directory path must be provided.
- The target file path must be writable and include the correct archive file extension.
- Authentication credentials must be valid and have appropriate permissions.
- Optional flags such as deletion and overwriting must be explicitly set based on the intended outcome.

Figure 22: Success skill example from Qwen3-32B: Tool Use

Success Skill Example: Evid. Grounding & Output Constr.

axis_name:
Evidence Grounding & Output Construction

skill_type:
principle

applicability:
Tasks requiring precise grounding of a target object from user data, filtering based on structured metadata, and constructing outputs that align with system constraints such as uniqueness, error handling, or aggregation.

grounding_principle:
Ground the target object by retrieving relevant data sources, filtering using verified schema or API fields, and ensuring alignment with the task's scope and constraints.

output_alignment_rule:
Construct outputs by extracting and processing identifiers or values from the grounded data, applying deduplication, error handling, or aggregation as needed to meet system or user requirements.

failure_to_avoid:
Failing to preserve fidelity in output construction by ignoring uniqueness constraints, misinterpreting schema fields, or omitting error handling for redundant or already-processed entities.

Figure 23: Success skill example from Qwen3-32B: Evidence Grounding & Output Construction

Success Skill Example: Task Understanding

axis_name:
Task Understanding

skill_type:
principle

applicability:
Tasks requiring selective action on data items based on composite conditions involving temporal, relational, and contextual filters.

core_principle:
Correctly interpret the task goal as a logical intersection of constraints—where action is scoped only to items satisfying all specified conditions—rather than as a collection of independent filters or a broad categorical match.

critical_constraints:
- Actions must be applied exclusively to items that meet all defined criteria, including temporal bounds, relationship-based eligibility, and data provenance.
- Relationship labels (e.g., 'roommates', 'family') must be resolved through structured, trusted sources of entity metadata rather than assumed or inferred connections.

misread_to_avoid:
Interpreting relational or temporal conditions loosely, such as acting on items that meet only one criterion or expanding scope based on proximity or partial matching.

Figure 24: Success skill example from Qwen3-235B: Task Understanding

Success Skill Example: Strategy Planning

axis_name:
Strategy Planning

skill_type:
workflow_steps

applicability:
When accessing and analyzing data from external, paginated, or distributed APIs to identify target values (e.g., extreme or frequent values) within user-specific collections.

workflow_steps:
- Discover available APIs and assess accessible endpoints, authentication mechanisms, and data structure
- Securely retrieve credentials and authenticate to obtain valid access tokens
- Retrieve primary data containers using pagination to ensure complete enumeration
- Iterate through containers to access individual items or records comprehensively
- Fetch detailed attributes for each item using granular API calls
- Aggregate and analyze item-level data to identify target results through comparison or computation

critical_checkpoints:
- Authentication is established before any data access
- Pagination is fully managed to prevent data omission
- All data is aggregated and processed in full before analysis
- Value identification is based on accurate and consistent computation

Figure 25: Success skill example from Qwen3-235B: Strategy Planning

Success Skill Example: Tool Use

axis_name:
Tool Use

skill_type:
tool_usage_steps

applicability:
When authenticating to a protected service using credential-based login to obtain an access token for authorized API interactions.

tool_usage_steps:
- Securely retrieve valid credentials from an authorized source without exposing secrets in plaintext
- Invoke the authentication tool with properly formatted credentials to obtain an access token
- Extract the access token from the successful authentication response for use in subsequent authorized API calls

critical_preconditions:
- Valid credentials must be available and correctly structured for the authentication endpoint; secrets must not be hardcoded or transmitted insecurely
- The authentication response must return an access token in a consistent, extractable format to enable further authorized interactions

Figure 26: Success skill example from Qwen3-235B: Tool Use

Success Skill Example: Evid. Grounding & Output Constr.

axis_name:
Evidence Grounding & Output Construction

skill_type:
principle

applicability:
When identifying a target object from a data collection based on a quantitative extremal criterion (e.g., maximum, minimum, highest frequency) and constructing a precise output from raw source data.

grounding_principle:
Ensure complete and systematic retrieval of all relevant records, applying exhaustive evaluation across the full dataset to accurately identify the target instance according to the specified quantitative criterion.

output_alignment_rule:
Construct the output by directly extracting the required value from the source record of the identified target without transformation, interpretation, or embellishment, preserving the original data's form and fidelity.

failure_to_avoid:
Premature filtering or incomplete aggregation that risks omitting the true extremal instance, and introducing any formatting, summarization, or modification not present in the source record during output construction.

Figure 27: Success skill example from Qwen3-235B: Evidence Grounding & Output Construction