RECURSIVE EXPERIENTIAL–WORKING MEMORY EVOLUTION FOR LONG-HORIZON AGENT HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement (RSI) remains hard in long-horizon tasks, where growing histories obscure the task state and misalign skill invocation, while failures are difficult to trace to specific harness components, hindering effective evolution. We introduce **Recuris, a recursive Experiential–Working Memory architecture for long-horizon agent harnesses**, in which Working Memory tracks task progress and guides skill selection from Experiential Memory, grounding skill use in current needs rather than the full history. This coupling also turns execution into structured evidence that localizes failures to specific memory components. Across tasks, a fixed Meta-Agent uses this evidence to make localized, validation-gated updates to the memory system. These updates reshape execution and yield new evidence, forming a **bounded recursive memory-evolution loop**. Across four challenging long-horizon benchmarks and ten models, Recuris substantially improves task success for **both frontier and open-source models**: on τ^2-Bench Retail it adds $+\mathbf{17.8}$ points to **GPT-5.6 Sol** and $+\mathbf{15.6}$ to **Claude Opus 5**, taking Opus 5 to $\mathbf{87.9}\%$, and $+16.6$/$+13.5$ points on Qwen3.6-27B/35B on SkillFlow. The advantage widens as the interaction horizon grows, to $+32.2$ points on the longest tasks, and common long-horizon failures fall by up to 80%. These results position recursively evolving memory as a scalable foundation for RSI, enabling agents to continuously transform accumulated experience into increasingly effective long-horizon behavior.

1 INTRODUCTION

Large language model (LLM) agents have demonstrated strong capabilities in reasoning, tool use, and autonomous execution (Yao et al., 2023; Qiu et al., 2025; Zhai et al., 2025). Increasingly, these capabilities are mediated by an **agent harness** (Lin et al., 2026a; Fan et al., 2026): an external execution layer that coordinates memory, skill invocation, task-state tracking, tool interaction, and verification around the underlying model (Zhou et al., 2026; Zhang et al., 2026a; 2025; Shang et al., 2025; Karten et al., 2026). Yet achieving **recursive self-improvement (RSI)** (Zhang et al., 2026c; Schmidhuber, 2003; Zweiger et al., 2025; Srikanth et al., 2026) remains a fundamental challenge, particularly in **long-horizon tasks** where goals, observations, and failures continuously evolve throughout execution (Wang et al., 2026b). In these settings, a capable harness must do more than provide a fixed set of tools or memories: it must maintain an accurate understanding of the current task state (Rozanov & Rei, 2025; Kim et al., 2025; Wu et al., 2024), invoke the appropriate experience as new problems emerge, and turn execution failures into improvements that affect future behavior (Yang et al., 2026b; Zhang et al., 2026e; Jiang et al., 2026b; Li et al., 2026c; Che et al., 2026). As interaction histories grow, however, agents often lose track of unresolved goals and invoke skills that are no longer aligned with the current task state (Wu et al., 2026b), and when a run fails, the long history makes it hard to tell which part of the harness is responsible (Liu et al., 2026a; Zhu et al., 2025; Shah, 2026; Zhang et al., 2026b). Long-horizon execution and evolution therefore pose two linked challenges for RSI: keeping skill invocation aligned with the task state as it changes, and identifying which component need improvement when execution fails (Shi et al., 2026).

Existing experiential-memory methods (Wang et al., 2026a; Ouyang et al., 2026; Zhang et al., 2026d; Fang et al., 2026) attempt to address the first challenge by retrieving skills either once from the initial instruction or repeatedly from the full interaction history, or by leaving the model to decide on its own which skill to use and when to invoke (Liu et al., 2026c). However, such retrieval becomes

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.

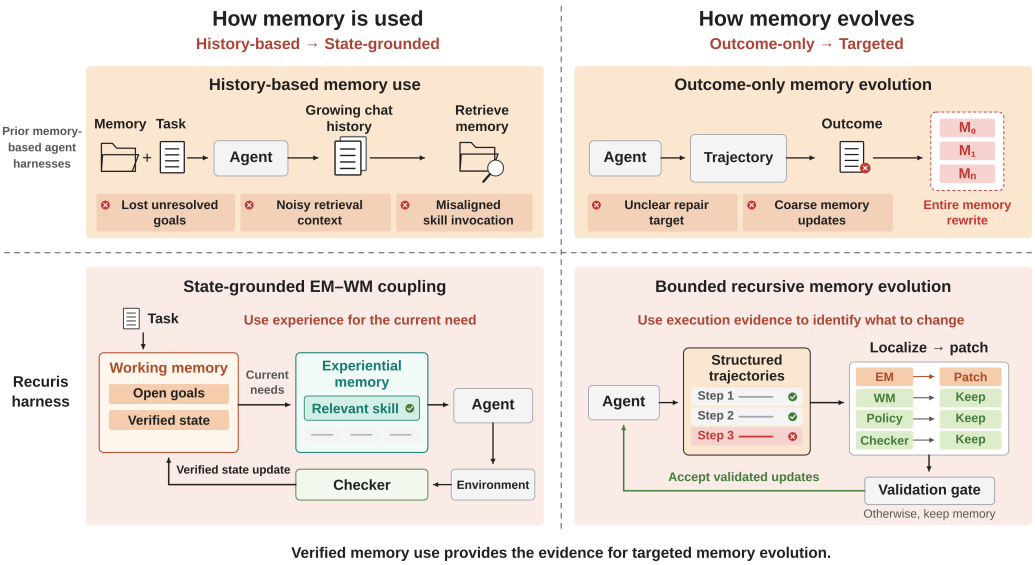


Figure 1: **Recuris makes memory use state-grounded and memory evolution targeted.** Top: history-based experience retrieval and outcome-based memory updating. Bottom: Recuris invokes skills according to a verified working state (left) and patches only the memory component that a structured trace implicates, subject to a validation gate (right).

increasingly unreliable as execution unfolds: the initial instruction may no longer reflect the current problem (Zhang et al., 2026g), while the growing context mixes outdated information, completed steps, unresolved requirements, and execution noise. Leaving the decision to the model does not solve this either, because it has no reliable signal of when a skill is needed. The central limitation is therefore not simply the absence of useful experience, but the lack of a **compact and reliable task state** that can continuously align stored experience with the agent's current execution needs.

We argue that **Working Memory (WM)** (Wu et al., 2026a; Chen et al., 2026) provides this missing state. WM tracks the agent's progress and unresolved goals and guides skill invocation from **Experiential Memory (EM)**; environment feedback then verifies progress before WM is updated, closing the loop **Task State** $\rightarrow$ **Skill Invocation** $\rightarrow$ **Execution Feedback** $\rightarrow$ **Updated Task State**. This **Experiential–Working Memory Coupling** keeps skill use grounded in the current task rather than the entire history, and it also provides a structured substrate for **memory evolution**. Existing approaches often learn mainly from final success or failure, which reveals that something went wrong but not whether the repair lies in the stored skill, the tracked state, the invocation mechanism, or the progress verifier. By linking task states, invoked skills, actions, and outcomes, the coupling turns execution into structured evidence, so a failure can be localized to one memory component and repaired without rewriting the entire memory: **EM–WM Coupling** $\rightarrow$ **Structured Evidence** $\rightarrow$ **Failure Localization** $\rightarrow$ **Targeted Memory Evolution**. Figure 1 contrasts both shifts with prior memory-based harnesses.

Based on this insight, we propose **Recuris**, a recursive Experiential–Working Memory architecture for long-horizon agent harnesses. Within each task, Recuris uses WM to maintain a verified task state and uses that state to invoke the most appropriate skill from EM. Across tasks, a fixed *Meta-Agent* analyzes the resulting structured execution traces, identifies the memory component most likely to provide an effective repair, and proposes a localized update. Candidate updates are incorporated only after passing a fixed validation process.

The updated Skill Memory then changes how the harness behaves on future tasks, producing new states, skill invocations, and execution evidence that can support subsequent updates. Recuris therefore forms a **bounded, validation-gated recursive loop**: memory shapes harness behavior, harness behavior produces diagnostic evidence, and this evidence is used to evolve the memory mechanisms that shape future behavior. The underlying LLM and outer improvement procedure remain fixed; recursion occurs within the externalized memory-control layer of the agent harness.

We evaluate Recuris on four long-horizon benchmarks covering tool-use dialogue, lifelong skill reuse, and challenging terminal tasks, across ten models from a 3B open-source model to frontier models. Recuris improves task success in 38 of the 40 model–benchmark pairs. On τ^2-Retail it improves **GPT-5.6 Sol** from 58.3 to 76.1 and **Claude Opus 5** from 72.4 to $\mathbf{87.9}$; on SkillFlow it improves Qwen3.6-27B from 42.2 to 58.7 and Qwen3.6-35B from 35.3 to 48.8. Frontier models are therefore not saturated on long-horizon tasks.

Our analyses locate where the gain comes from. The advantage grows with the interaction horizon instead of decaying, reaching $+32.2$ points on the longest tasks, and common long-horizon failures fall by up to 80%. Localizing a failure to the responsible memory component is far more accurate from the structured trace than from the task outcome alone, 64.8% against 13.0%, which is what makes a scoped repair possible. Because recursion is confined to the memory-control layer, every gain is obtained with the base model left exactly as its provider shipped it.

Our contributions are summarized as follows:

- We establish **state-grounded memory use** as a key requirement for recursive self-improvement in long-horizon agent harnesses, where evolving task states demand continual alignment between accumulated experience and current execution needs.
- We introduce **Recuris, a recursive Experiential–Working Memory architecture** that couples persistent experience with dynamically maintained and evidence-grounded task states, enabling adaptive skill invocation throughout long-horizon execution.
- We show how EM–WM coupling turns harness execution into **structured diagnostic evidence**, enabling component-level failure localization and targeted evolution of the Skill Memory mechanisms that govern future behavior.
- We evaluate Recuris on four long-horizon benchmarks and ten models, studying execution reliability, skill invocation, failure localization, evolution stability, and cross-task transfer of evolved memory, and we observe consistent gains across model scale, including $+\mathbf{17.8}$ points for **GPT-5.6 Sol** and $+\mathbf{15.6}$ points for **Claude Opus 5** on τ^2-bench.

2 Related Work

Experiential memory and agent skills. Reusable experience (Chhikara et al., 2025; Xu et al., 2025) can take the form of executable skills in Voyager (Wang et al., 2024), workflows in AWM (Wang et al., 2025), or insights extracted from successful and failed trajectories in ExpeL (Zhao et al., 2024). Agent Skills packages procedures and supporting resources for on-demand loading (Anthropic, 2025). Beyond storing useful content, selection must account for execution context (Li et al., 2026b; Huang et al., 2026; Miao et al., 2026; Han et al., 2026; Hu et al., 2026): AutoGuide generates context-conditioned guidelines (Fu et al., 2024), while SGDR matches sub-procedures to the task goal and current webpage state (Li et al., 2026a). Recuris instead couples reuse to an explicitly maintained, verified working state, whose confirmed progress and unresolved requirements determine which experience is relevant at execution events.

Working memory for LLM agents. Cognitive architectures distinguish working memory from persistent experience (Sumers et al., 2024). Magentic-One maintains task and progress ledgers to guide execution (Fourney et al., 2024); StructAgent regulates task progress through verifier-backed state transitions and supports checkpointing and recovery (Wu et al., 2026a). Recuris builds on such state maintenance but separates proposed from committed progress, requiring execution evidence before accepting a completion claim. The verified state then serves two roles: it guides retrieval from procedural memory during execution, and its transitions become evidence for diagnosing memory failures across tasks.

Recursive self-improvement and memory evolution. Self-improvement can modify an improvement program itself (Fernando et al., 2024; Yin et al., 2025; Robeyns et al., 2025), as in STOP (Zelikman et al., 2024), or retain the model and refine external experience (Zhou et al., 2025; Liu et al., 2026b; Yang et al., 2026c; Jiang et al., 2026a; Suzgun et al., 2026). Reflexion stores textual feedback (Shinn et al., 2023); ACE incrementally curates strategy playbooks (Zhang et al., 2026e); SkillOpt turns scored rollouts into bounded skill-document edits with validation-based acceptance

(Yang et al., 2026b). Other approaches refine hierarchical skill libraries (SkillX; Wang et al., 2026a), compile online execution contexts into reusable harness skills (Evo-Harness; Wei et al., 2026), or manage skill authoring and retirement (Ratchet; Zhang et al., 2026f). Recuris instead evolves a coupled memory system of experience, working-state specifications, invocation policies, and checkers, using structured traces to diagnose failures and patch the implicated components; admitted memories shape the evidence of later rounds, while the model and the improvement procedure stay fixed.

3 METHOD

Across benchmarks, Recuris uses the same Skill Memory structure and within-task EM–WM coupling inside a fixed agent harness (Figure 2 and Section 3.2); a fixed Meta-Agent iteratively updates the memory (Sections 3.3 and 3.4).

3.1 SKILL MEMORY: AN EVOLVABLE MEMORY-CONTROL LAYER

The harness runs a frozen LLM π_θ with tools $\mathcal{T}$. At step t of a task x, the agent emits an action a_t (a message or a tool call) and receives an observation o_t (a user reply or a tool result). With a Skill Memory $\mathcal{M}$, a run on x yields a structured trace Γ (Section 3.2) and a reward $R(x, \Gamma) \in [0, 1]$. In the held-out cross-task setting, we seek memory that performs well on unseen tasks,

$$\max_{\mathcal{M}} \; J(\mathcal{M}) = \mathbb{E}_{x \sim \mathcal{D}_{\text{test}}} \, \mathbb{E}_{\Gamma \sim \text{Run}(x;\, \pi_\theta, \mathcal{M})} \big[R(x, \Gamma) \big], \tag{1}$$

while π_θ, $\mathcal{T}$, and the evolution procedure stay fixed and only an evolution split $\mathcal{D}_{\text{evolve}}$ and a development split $\mathcal{D}_{\text{dev}}$ provide feedback. At evolution round k, the memory has four declarative components:

$$\mathcal{M}_k = (\mathcal{E}_k, \mathcal{W}_k, \rho_k, \mathcal{C}_k), \qquad \mathcal{E}_k = \{(\kappa_i, s_i)\}_i, \qquad \rho_k = \{(\phi_m, \psi_m)\}_m, \qquad \mathcal{C}_k = \{\chi_g\}_g. \tag{2}$$

The **experiential memory** $\mathcal{E}_k$ stores skills s_i (Anthropic, 2025), such as procedures, domain rules, and notes on how past actions resolved, indexed by keys κ_i. The **working-memory specification** $\mathcal{W}_k$ defines the task state $w_t = \{(g, \sigma_t(g), v_t(g), b_t(g))\}_g$, in which each goal g has a status $\sigma_t(g) \in \{\texttt{pending}, \texttt{done}, \texttt{blocked}\}$, evidence $v_t(g)$, and an optional blocker $b_t(g)$, together with the instructions for proposing updates. The **invocation policy** ρ_k contains invocation rules that pair a trigger predicate ϕ_m over the state and execution events with a key function ψ_m. The **checkers** $\mathcal{C}_k$ are completion predicates χ_g that test a proposed change against evidence such as a structured tool receipt. A patch can rewrite skill text, state fields, triggers, keys, or predicates, while the runtime that executes them stays fixed; $\mathcal{M}_k$ carries these changes into later rounds.

3.2 WITHIN A TASK: VERIFIED EM–WM COUPLING

The harness initializes w_0 under $\mathcal{W}_k$ and renders it at every step, so unresolved goals stay visible as the history h_t grows.

State-grounded skill invocation. At an execution event e_t, each triggered invocation rule retrieves matching skill guidance into the agent's context:

$$\mathcal{E}_t = \bigcup\nolimits_{(\phi,\psi) \in \rho_k :\, \phi(w_t, e_t) = 1} \big\{ s_i \in \mathcal{E}_k : \kappa_i = \psi(w_t, e_t) \big\}, \qquad a_t \sim \pi_\theta(\cdot \mid x, h_t, w_t, \mathcal{E}_t). \tag{3}$$

For example, in the τ^2 domains, the event is a drafted state-changing call $\hat{a}_t \sim \pi_\theta(\cdot \mid x, h_t, w_t)$ and the key is its tool name: the harness withholds $\hat{a}_t$, returns a synthetic not-executed result, and invokes the skills in $\mathcal{E}_t$ before the agent writes the call that actually runs. Because the draft is conditioned on w_t, skill invocation follows the goal the agent is currently pursuing.

Verified state update. After $o_t = \text{Env}(a_t)$, the updater proposes $\widetilde{w}_{t+1} = U_{\mathcal{W}_k}(w_t, a_t, o_t)$, and a fixed kernel commits each goal only as far as the evidence supports:

$$\sigma_{t+1}(g) = \begin{cases} \texttt{done}, & \chi_g(\widetilde{w}_{t+1}, a_t, o_t) = 1, \\ \widetilde{\sigma}_{t+1}(g), & \widetilde{\sigma}_{t+1}(g) \neq \texttt{done}, \\ \sigma_t(g), & \text{otherwise.} \end{cases} \tag{4}$$

A tool call, a loaded skill, or a claim of success therefore cannot mark a goal `done`. In Figure 2(a), the tool acknowledges an exchange that reuses the original item ID; the checker finds

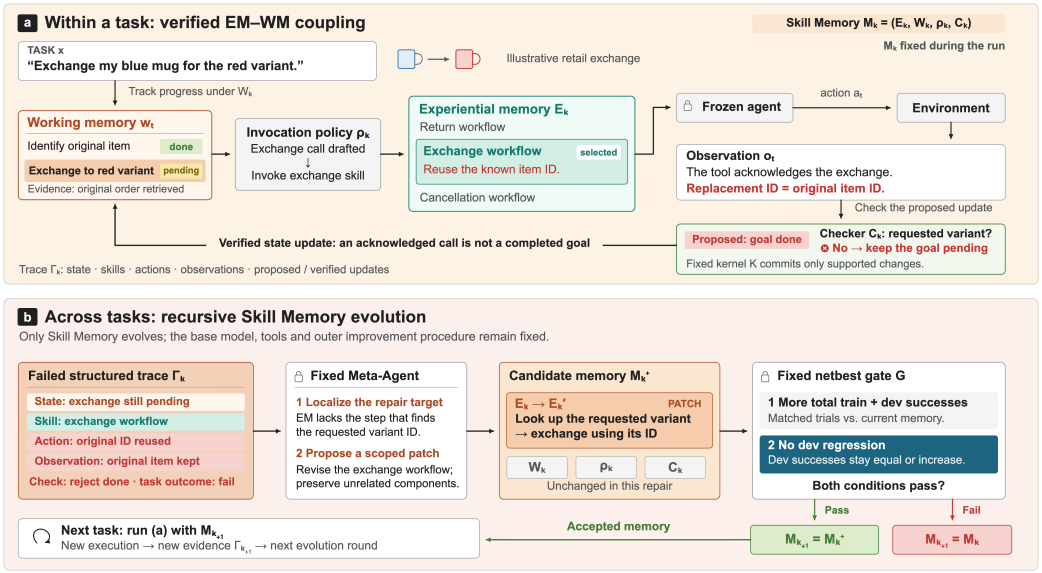


Figure 2: **Overview of Recuris on an illustrative Retail exchange.** (a) With the exchange goal pending, the drafted exchange call triggers invocation of the matching skill; since the replacement reuses the original item ID, the checker keeps the goal pending. (b) The Meta-Agent traces the failure to $\mathcal{E}$ and patches only the exchange workflow; the netbest gate accepts the patch when combined evolution and dev successes increase without a decrease on dev, and the next round starts from the admitted memory (admission settings in Appendix A.2).

the requested variant missing and keeps the goal pending. Each run logs a **structured trace** $\Gamma_k = \big(x, \{(w_t, \mathcal{E}_t, a_t, o_t, \widetilde{w}_{t+1}, c_t, w_{t+1})\}_{t=1}^{L}, y\big)$ with checker decisions c_t and outcome y. Unlike a raw trajectory $\{(a_t, o_t)\}_t$, it ties every skill invocation to the state that triggered it and separates proposed from committed updates.

3.3 Across Tasks: Bounded Recursive Self-Improvement

We describe cross-task evolution using the split-based τ^2 setting, where a fixed Meta-Agent $\mathcal{A}$ improves $\mathcal{M}_k$ from the failed traces of $\mathcal{D}_{\text{evolve}}$.

Diagnose and patch. The Meta-Agent assigns each observed failure f_j to the component z_j whose repair should remove it, and edits only the implicated components $Z_k = \{z_j\}_j$:

$$D_k = \mathcal{A}(\Gamma_k, \mathcal{M}_k) = \{(f_j, z_j)\}_{j=1}^{J_k}, \qquad \mathcal{M}_{k,z}^{+} = \begin{cases} \text{Apply}(\mathcal{M}_{k,z}, \Delta m_z), & z \in Z_k, \\ \mathcal{M}_{k,z}, & z \notin Z_k, \end{cases} \tag{5}$$

where $\mathcal{M}_{k,z}$ is component $z \in \{\mathcal{E}, \mathcal{W}, \rho, \mathcal{C}\}$ and Δm_z is the Meta-Agent's edit to it. Missing or incorrect guidance points to $\mathcal{E}$, an omitted goal or unsuitable update to $\mathcal{W}$, a missed, late, or irrelevant skill invocation to ρ, and a wrong acceptance or rejection to $\mathcal{C}$; the trace makes these cases distinguishable. In Figure 2(b), the failed exchange is traced to $\mathcal{E}$ and only the exchange workflow is revised.

Admit and recurse. A fixed netbest gate $\mathcal{G}$ compares the candidate with the incumbent on all tasks in $\mathcal{D}_{\text{evolve}}$ and $\mathcal{D}_{\text{dev}}$. One round is thus an operator Φ, and the loop applies it to its own output:

$$\mathcal{M}_{k+1} = \Phi(\mathcal{M}_k) = \begin{cases} \mathcal{M}_k^{+}, & \mathcal{G}(\mathcal{M}_k^{+}, \mathcal{M}_k) = 1, \\ \mathcal{M}_k, & \text{otherwise}, \end{cases} \qquad \mathcal{M}_K = \Phi^K(\mathcal{M}_0). \tag{6}$$

Netbest admits a valid candidate when $\Delta n_{\text{evolve}} + \Delta n_{\text{dev}} > 0$ and $\Delta n_{\text{dev}} \geq 0$, where Δn is the candidate-minus-incumbent change in successful task trials (Appendix A.2). Held-out tasks never enter diagnosis, patching, or admission. The loop is *recursive* because round $k+1$ diagnoses traces produced under $\mathcal{M}_{k+1}$, so each admitted patch changes both later behavior and the evidence for the next repair; restarting every round from $\mathcal{M}_0$ would discard both. It is *bounded* because π_θ, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{G}$, and the patch interface stay fixed; only the Skill Memory changes.

3.4 Test-Time Adaptation

Cross-task evolution relies on experience that recurs across tasks, such as policies within a τ^2 domain or procedures within a SkillFlow family. Terminal-Bench 2.1 offers little such reuse: each task has its own environment, toolchain, and goal (Merrill et al., 2026). We therefore run the same loop within each task. Starting from a seed memory, the Meta-Agent diagnoses each unsuccessful attempt from its trajectory and outcome and updates the four-component Skill Memory for the next attempt; it sees no hidden tests or reference outputs. With no other tasks to validate against, this mode has no development gate. Retrying with fixed memory uses the same success signal, so we compare the two at the same attempt limit to separate adaptation from retries (Appendix C.1).

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on τ^2-Retail (114 tasks) and τ^2-Airline (50 tasks) for policy-constrained tool dialogue (Barres et al., 2026), SkillFlow (166 tasks in 20 families) for procedural skill reuse (Zhang et al., 2026h), and Terminal-Bench 2.1 for terminal tasks (Merrill et al., 2026). A task counts as solved only when it receives the full benchmark reward, and we report task-averaged success (avg@k) in percent. Cohorts and repetition counts are listed in Appendix A.

Models. We evaluate ten frozen execution models: six open-source models (3B–35B) and four frontier models. Among the latter, we choose Doubao-2.0-Pro for its substantial room for improvement. For each frozen-memory benchmark, its evolved Skill Memory is frozen and transferred unchanged to other frontier and open-source models to test generalization; this is *shared memory*. Open-source main results use *model-specific memory*, separately evolved from each model's own execution feedback. Doubao-2.1-Pro is the default Meta-Agent. The reference Retail split uses 16 evolution, 12 dev, and 86 held-out tasks. SkillFlow follows its official family-based evolution protocol; Terminal-Bench uses task-local adaptation (Section 3.4).

Baselines. First, we compare each model with its baseline under the corresponding benchmark harness: τ^2 tool calling, Qwen Code, or Terminus-2. Second, with the worker fixed, we compare Recuris with five recent self-evolving memory methods that evolve skills or context playbooks from experience, ACE (Zhang et al., 2026e), SkillX (Wang et al., 2026a), Ratchet (Zhang et al., 2026f), SkillOpt (Yang et al., 2026b), and Evo-Harness (Wei et al., 2026), each ported to the same worker and tasks.

4.2 Main Results

Full-domain τ^2 entries in Table 1 aggregate train, dev, and test; Appendix C.2 reports the subset results. Recuris improves long-horizon task success from a 3B-sized model to frontier models, where an agent must carry several pending requests or procedure steps through a long interaction. On Retail, Recuris improves GPT-5.6 Sol by 17.8 points and Claude Opus 5 by 15.6 points, with Opus 5 reaching 87.9% using the transferred memory. Even frontier models thus lose track of long tasks, and a memory-control layer recovers much of this loss without changing the model; test-time adaptation also improves all ten models on Terminal-Bench 2.1.

Transfer across models. Shared-memory transfer tests whether evolution learns useful procedures beyond the source model's behavior. Doubao-evolved memory improves the three other frontier models unchanged on Retail and SkillFlow (Table 1); Figure 3(a) further compares its transfer to open-source models with model-specific evolution. Each open-source model benefits more from its own evolved memory than from the shared memory. Together, these results suggest that Skill Memory captures both common task difficulties, such as dropped requests and policy constraints, and model-specific execution habits (Lin et al., 2026b; Yang et al., 2026a). Transfer exploits the former, while evolution on the target model additionally addresses the latter.

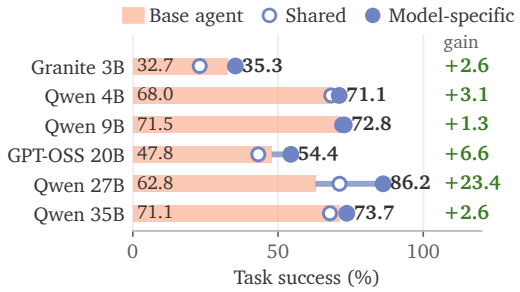
4.3 Comparison with Self-Evolving Memory Methods

With the worker fixed, Recuris outperforms ACE (Zhang et al., 2026e), SkillX (Wang et al., 2026a), Ratchet (Zhang et al., 2026f), SkillOpt (Yang et al., 2026b), and Evo-Harness (Wei et al., 2026) in

Table 1: **Task success (%) of the baseline and Recuris.** Small numbers are point changes before rounding (†: paired interval excludes zero). Frozen-memory entries use Doubao-evolved shared memory for frontier models and model-specific memory for open-source models. SkillFlow uses $k = 1$ for Sol, Opus, and Gemini. Retail 9B, GPT-OSS, and 35B report test-subset results.

	Frozen memory						*Test-time adaptation*	
	τ^2-**Retail**		τ^2-**Airline**		**SkillFlow**		**TB 2.1** (avg@4)	
Model	Base	Recuris	Base	Recuris	Base	Recuris	Base	Recuris
open-source models								
Granite-4.1-3B	32.7	**35.3** +2.6	38.5	**43.0** +4.5	**0.3**	0.0 −0.3	0.6	**3.1** +2.5
Qwen3.5-4B	68.0	**71.1** +3.1	75.3	**79.0** +3.8	6.0	**7.1** +1.1	10.1	**13.0** +2.9
Qwen3.5-9B	77.6	**79.6** +2.0	75.5	**78.5** +3.0	15.1	**18.4** +3.4	17.4	**20.5** +3.1
GPT-OSS-20B	45.4	**60.8** +15.4	54.8	**59.3** +4.5†	7.8	**10.4** +2.6†	3.9	**6.7** +2.8
Qwen3.6-27B	62.8	**86.2** +23.4†	79.0	**80.0** +1.0	42.2	**58.7** +16.6†	38.8	**42.1** +3.3
Qwen3.6-35B	78.2	**78.5** +0.3	80.3	**81.5** +1.3	35.3	**48.8** +13.5†	33.1	**36.4** +3.3
Frontier models								
Doubao-2.0-Pro	58.1	**83.5** +25.4	75.5	**84.0** +8.5	34.6	**51.4** +16.8†	46.1	**48.9** +2.9
Gemini 3.7 Flash	73.5	**78.3** +4.8	**86.5**	85.0 −1.5	49.4	**58.4** +9.0	79.8	**82.4** +2.6
GPT-5.6 Sol	58.3	**76.1** +17.8†	79.0	**86.0** +7.0†	50.0	**62.7** +12.7	83.2	**86.4** +3.2
Claude Opus 5	72.4	**87.9** +15.6†	89.5	**90.5** +1.0	52.4	**64.5** +12.0	84.6	**88.4** +3.8

(a) Base, shared, and model-specific memory



(b) Self-evolving memory methods

	τ^2-**Retail** 86 tasks × 4	τ^2-**Airline** 29 tasks × 4	**SkillFlow** 114 tasks × 3
Base agent	60.2	86.2	35.1
ACE	56.7	82.8	38.6
SkillX	60.5	84.5	35.7
Ratchet	57.3	83.6	38.0
SkillOpt	66.6	78.5	45.6
Evo-Harness	57.3	82.8	30.7
Recuris	**81.7**	**87.8**	**51.4**
Lead over best	**+15.1**	**+3.3**	**+5.8**

Figure 3: **Model-specific evolution improves every open-source model on Retail, and Recuris outperforms five self-evolving memory methods in all three domains.** (a) 114 Retail tasks; right column: gain of model-specific memory over the base agent. (b) Doubao-2.0-Pro worker; shading marks the change from the base agent, the bottom row the lead over the strongest other method; methods in release order.

all three domains, leading the strongest of them by 15.1, 3.3, and 5.8 points on Retail, Airline, and SkillFlow, and it is the only method that improves on the base agent everywhere (Figure 3(b)). The other methods are inconsistent: SkillOpt is the strongest of them on Retail and SkillFlow but the weakest on Airline, where all five fall below the base agent. One explanation is that useful memory depends on how a task uses it: Recuris couples reusable guidance with a verified working state that determines when it is needed. The following τ^2 ablation examines why this combination helps beyond keeping the full library in context.

4.4 EM–WM Coupling in Long-Horizon Tasks

Longer tasks, larger gains. If Recuris helps by keeping the task state explicit, its benefit should grow with interaction length. Grouping τ^2 tasks by the median number of turns they need, the base agent degrades on longer tasks while Recuris holds up, so the gain widens from 12.5 to 32.2 points (Figure 4(a)). The failures it removes point the same way: those of a lost or unverified state, such as hallucinated completions and omitted writes, fall by 62% to 86% (Figure 4(b)), while error cascades and missed reads, which depend less on the tracked state, fall by about a fifth.

Skills are effective only when guided by the task state. To locate these gains, we remove parts of one evolved package per domain (Figure 5): *EM only* invokes skills without a working state,

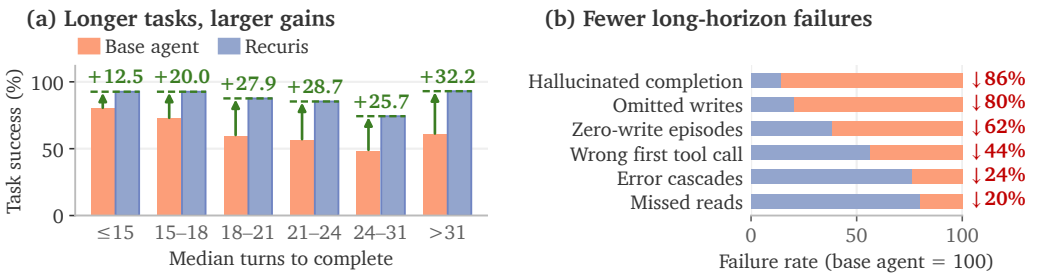


Figure 4: **Recuris gains more on longer tasks and reduces common long-horizon failures.** Doubao-2.0-Pro on τ^2-bench, avg@4. (a) Success by the median number of turns a task needs; (b) failure frequency relative to the base agent (= 100).

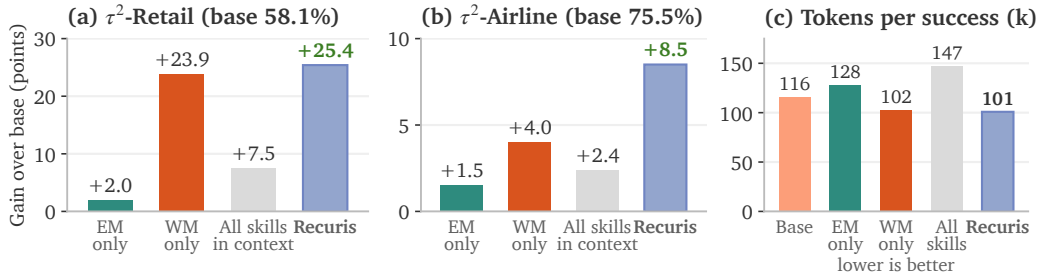


Figure 5: **EM–WM coupling gives the largest gain in both domains and the lowest token cost on Retail.** One evolved package per domain, Doubao-2.0-Pro, k=4: EM only omits the working state, WM only omits the skills, and all skills in context adds the whole library to the working state at every turn. (a, b) Gains over the base agent; (c) tokens per success.

WM only keeps the verified state without skills, and *all skills in context* keeps both but shows the whole library at every turn. Without a state, skills barely improve the base agent. Adding the whole library on top of the working state and letting the model choose is worse than adding nothing: it falls below WM only in both domains and 18.0 points below Recuris on Retail, at 46% more tokens per success. With the working state guiding action selection and relevant skills invoked before the drafted write, Recuris achieves the highest success in both domains. We attribute this to the length of the interaction: the relevant rule is one of many and the history no longer shows which goal is open, so an unselected library distracts more than it guides. The working state supplies the missing signal of which skill applies now, which is the role of EM–WM coupling.

Different tasks rely on different memory components. Tasks also differ in how much they need each kind of memory. On Airline, coupling doubles the gain of the working state alone, because its tasks turn on policy rules, such as which fares can be changed, that matter exactly when a goal meets such a constraint. On Retail, the working state alone yields nearly all of the gain, because most failures are tracked requests that are never executed, which a verified state catches without any skill. Removing single mechanisms confirms this split: dropping pre-write skill invocation costs 13.5 points on Airline, and hiding the working state costs 17.3 points on Retail. This contrast motivates adapting memory to the task, so Recuris leaves that adaptation to evolution: the Meta-Agent repairs whichever component the failures implicate, and the memory follows what each task requires.

4.5 Structured Diagnosis and Recursive Evolution

We now turn to the second challenge: attributing failures and improving memory across tasks.

Failure localization. We inject known faults into $\mathcal{E}$, $\mathcal{W}$, or ρ and ask the Meta-Agent model, Doubao-2.1-Pro, to name the faulty component from the outcome, the raw trajectory, or the structured trace Γ (Figure 6). The outcome alone is below chance, since it shows that a task failed but not where. Raw trajectories expose faulty skill content but never invocation faults, because whether the right skill arrived at the right step is not visible in the dialogue. Structured traces, which the coupling produces as it runs, record the state, invoked skills, and checker decisions at each step, im-

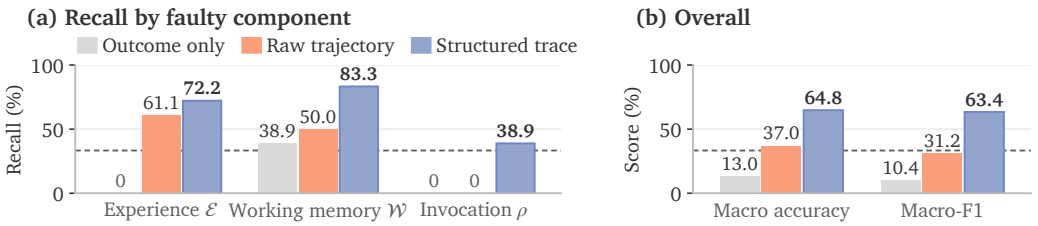


Figure 6: **Structured traces localize injected faults much more accurately than outcomes or raw trajectories.** Doubao-2.1-Pro names the faulty component from one evidence source, 54 judgments each: (a) recall per injected component, (b) macro accuracy and macro-F1; dashed line: chance.

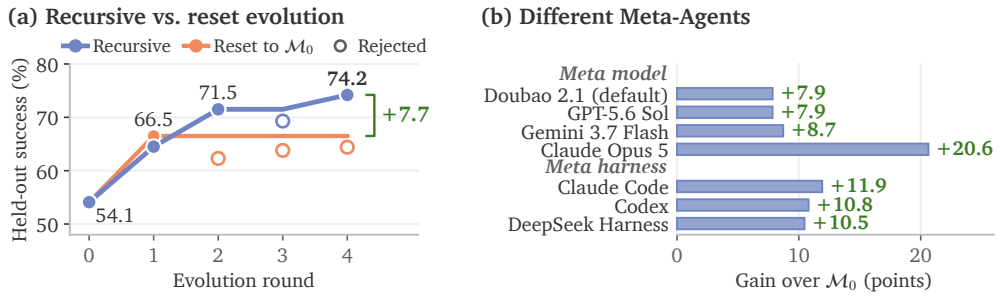


Figure 7: **Recursive evolution keeps improving where restarting from $\mathcal{M}_0$ stalls, and every tested Meta-Agent improves on the initial memory.** 86 held-out Retail tasks, Doubao-2.0-Pro worker, k=4. (a) Retained memory per round when patching the last admitted memory (recursive) or $\mathcal{M}_0$ (reset); hollow markers: rejected candidates. (b) Gains over each group's $\mathcal{M}_0$.

proving macro accuracy from 37.0% to 64.8%. Diagnosis is bounded by what the evidence records, so recording these decisions is what makes component-scoped repair possible.

Recursion, not repeated search, sustains the gains. To isolate recursion, we rerun the loop with the same rounds and gate but patch $\mathcal{M}_0$ in every round instead of the last admitted memory (Figure 7(a)). After one round, both arms hold a single patch on $\mathcal{M}_0$ and lie within 2.0 points; afterwards every reset candidate falls below that first patch and is rejected, while the recursive loop keeps improving to 74.2%, 7.7 points above reset. This suggests that retaining repairs changes the failures available to later rounds: moving past one failure can expose subsequent problems and let new patches build on earlier fixes. Resetting the memory instead revisits the initial system and prevents patches from accumulating. The comparison therefore supports learning from the behavior of the previous improvement, which is the loop Recuris closes.

Gains do not depend on a particular Meta-Agent. With the worker, evidence format, patch interface, and gate fixed, every tested Meta model and Meta harness improves held-out success, by 7.9 to 20.6 points (Figure 7(b)), and the three harnesses lie within 1.5 points of one another. Because each Meta-Agent diagnoses structured traces through the same interface and may only make gated edits to the implicated components, even a moderate model finds patches that pass; a stronger model finds better ones, as Claude Opus 5 gains 20.6 points while the other three Meta models gain 7.9 to 8.7. The gains thus come from the loop, and a stronger Meta-Agent improves the memory further without changing the model that executes the task.

5 Conclusion

Recuris couples experiential memory with a verified working state, so that skills are used when the task state calls for them and every execution leaves component-level evidence, which a fixed Meta-Agent turns into scoped memory updates in a bounded recursive loop. Across ten models and four benchmarks, it improves long-horizon success, gains most on the longest tasks, and keeps improving where restarting stalls.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing, help us discover and find related work, and help write plotting and evaluation scripts. The research ideas, the design of Recuris, and its implementation were developed by the authors. We reviewed and verified all AI-assisted text and code; we take responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work evaluates agents on public benchmarks with simulated users and environments; it involves no human subjects and no personal data. Agents that revise their own memory could acquire undesirable procedures. Recuris confines every change to a declarative memory layer that can be inspected, admits cross-task updates through a fixed validation gate, and leaves the model and tools unchanged.

REPRODUCIBILITY STATEMENT

Section 3 defines the Skill Memory components, the invocation and state-update rules, the patch interface, and the admission rules. Appendix A specifies the benchmarks, task splits, repetition counts, execution and user-simulator models with their reasoning settings, and the evaluation metrics. Appendix B illustrates the four memory components; Appendix C separates initialization from evolution, and Appendix D walks through execution and an admitted update. The fault-injection and test-time adaptation protocols are in Appendices A.3 and C.1. We will release the code, the evolved Skill Memory packages, and the evaluation scripts upon publication.

REFERENCES

Anthropic. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

Lirong Che, Yuzhe Yang, Peiwen Lin, Chuang Wang, Xueqian Wang, and Jian Su. DemoEvolve: Overcoming sparse feedback in agentic harness evolution with demonstrations, 2026. URL https://arxiv.org/abs/2605.24539.

Yaoqi Chen, Haibin Lai, Yuru Feng, Chuyu Han, Qianxi Zhang, Baotong Lu, Menghao Li, Xinjiang Wang, Zhirui Wang, Shusen Xu, Zengzhong Li, Zewen Jin, Hao Wu, Cheng Li, and Qi Chen. Beyond Semantic Organization: Memory as Execution State Management for Long-Horizon Agents, 2026. URL https://arxiv.org/abs/2606.06090.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI)*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160.

Run-Ze Fan, Zihao Zhang, Simin Ma, Yebowen Hu, Shouju Wang, Kaiqiang Song, Fei Liu, Hamed Zamani, and Xiaoyang Wang. An empirical study of harness design for coding agents, 2026. URL https://arxiv.org/abs/2609.20804.

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

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024.

Adam Fourney et al. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, pp. 119919–119948, 2024.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. LEGOMem: Modular procedural memory for multi-agent LLM systems for workflow automation. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 3116–3118. International Foundation for Autonomous Agents and Multiagent Systems, 2026. doi: 10.65109/VLUA1303. Extended abstract.

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

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From Raw Experience to Skill Consumption: A Systematic Study of Model-Generated Agent Skills. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026. URL https://openreview.net/forum?id=GciYMFDFBW.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. XSkill: Continual learning from experience and skills in multimodal agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, 2026a. URL https://openreview.net/forum?id=AjP1yvCyoG.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. Harnessevolve: Learning from reference trajectories for reliable agent self-evolution, 2026b. URL https://arxiv.org/abs/2609.00829.

Seth Karten, Joel Zhang, Tersoo Upaa, Jr., Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual Harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33433–33465, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1697. URL https://aclanthology.org/2025.emnlp-main.1697/.

Jiaxi Li et al. Online skill learning for web agents via state-grounded dynamic retrieval, 2026a.

Yanchao Li, Wanhao Liu, Ben Gao, Jiaqing Xie, Zhehong Ai, Na Zou, Yuqiang Li, and Tianfan Fu. SkillsInjector: Dynamic Skill Context Construction for LLM Agents, 2026b. URL https://arxiv.org/abs/2605.29794.

Yanzhou Li, Yiran Zhang, Xiaoyu Zhang, Xiaoxia Liu, and Yang Liu. CODESKILL: Learning self-evolving skills for coding agents, 2026c. URL https://arxiv.org/abs/2605.25430.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026a. URL https://arxiv.org/abs/2604.25850.

Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents, 2026b. URL https://arxiv.org/abs/2605.30621.

Jiale Liu et al. Who&When Pro: Can LLMs Really Attribute Failures in AI Agents? arXiv preprint arXiv:2607.09996, 2026a.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Evolvemem: Self-evolving memory architecture via autoresearch for llm agents, 2026b.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. How Well Do Agentic Skills Work in the Wild: Benchmarking LLM Skill Usage in Realistic Settings. In *Conference on Language Modeling (COLM)*, 2026c.

Mike A. Merrill et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Yongliang Miao, Ziyang Yu, Liang Zhao, Bowen Zhu, and Hasibul Haque. SkillLens: Adaptive Multi-Granularity Skill Reuse for Cost-Efficient LLM Agents, 2026. URL https://arxiv.org/abs/2605.08386.

Siru Ouyang, Jun Yan, I-Hung Hsu, et al. ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory. In *International Conference on Learning Representations (ICLR)*, 2026.

Jiahao Qiu et al. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025.

Maxime Robeyns et al. A self-improving coding agent. In *ICLR 2025 Workshop on Scaling Self-Improving Foundation Models (SSI-FM)*, 2025. arXiv:2504.15228.

Nikolai Rozanov and Marek Rei. Stateact: Enhancing llm base agents via self-prompting and state-tracking. In *REALM Workshop @ ACL 2025*, 2025. arXiv:2410.02810.

Jürgen Schmidhuber. Gödel machines: Self-referential universal problem solvers making provably optimal self-improvements. *arXiv preprint cs/0309048*, 2003.

Jaineet Shah. Causal agent replay: Counterfactual attribution for llm-agent failures, 2026. URL https://arxiv.org/abs/2606.08275.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *International Conference on Learning Representations (ICLR)*, 2025.

Zitong Shi, Yixuan Tang, and Anthony Kum Hoe Tung. A-TMA: Decoupling State-Aware Memory Failures in Long-Term Agent Memory, 2026. URL https://arxiv.org/abs/2607.01935.

Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023. arXiv:2303.11366.

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents, 2026. URL https://arxiv.org/abs/2609.26457.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. arXiv:2309.02427.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic Cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, March 2026. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

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, 2026a. URL https://arxiv.org/abs/2604.04804.

Guanzhi Wang et al. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. arXiv:2305.16291.

Xinyu Jessica Wang et al. The long-horizon task mirage? diagnosing where and why agentic systems break, 2026b.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents, 2026c. URL https://arxiv.org/abs/2607.12227.

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

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-Harness: Context-to-Harness Skill Compilation for Self-Evolving Agents, 2026. URL https://arxiv.org/abs/2608.15071.

Wenyi Wu et al. Structagent: Harness long-horizon digital agents with unified causal structure, 2026a.

Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. Remember When It Matters: Proactive Memory Agent for Long-Horizon Agents, 2026b. URL https://arxiv.org/abs/2607.08716.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. StateFlow: Enhancing LLM task-solving through state-driven workflows. In *Conference on Language Modeling (COLM)*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, pp. 17577–17604, 2025. doi: 10.52202/085713-0593.

Siqi Yang, Qianlan Yang, Yu-Xiong Wang, Saurabh Pujar, and Martin Hirzel. One recipe, many harnesses: What self-evolution encodes across languages and models, 2026a. URL https://arxiv.org/abs/2608.10178.

Yifan Yang et al. Skillopt: Executive strategy for self-evolving agent skills, 2026b.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. AutoSkill: Experience-Driven Lifelong Learning via Skill Self-Evolution, 2026c. URL https://arxiv.org/abs/2603.01145.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. In *Conference on Language Modeling (COLM)*, 2024.

Yunpeng Zhai et al. Agentevolver: Towards efficient self-evolving agent system, 2025.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, Chuanrui Hu, Yafeng Deng, and Shuicheng Yan. JIT-Agent: Scaling Harness Intelligence via Just-in-Time Harness Evolution, 2026a. URL https://arxiv.org/abs/2608.25593v2.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations (ICLR)*, 2026b. URL https://openreview.net/forum?id=l05DseqvuD.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-Harness: Harnesses That Improve Themselves, 2026c. URL https://arxiv.org/abs/2606.09498v3.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026d. URL https://openreview.net/forum?id=3R1dZaZmxU.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, et al. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. In *International Conference on Learning Representations (ICLR)*, 2026e.

Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Ratchet: A Minimal Hygiene Recipe for Self-Evolving LLM Agents, 2026f. URL https://arxiv.org/abs/2605.22148v1.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. AdaMEM: Test-time adaptive memory for language agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026g.

Ziao Zhang et al. Skillflow: Benchmarking lifelong skill discovery and evolution for autonomous agents, 2026h.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Chenyu Zhou et al. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering, 2026.

Huichi Zhou et al. Memento: Fine-tuning llm agents without fine-tuning llms, 2025. v1 titled AgentFly.

Kunlun Zhu et al. Where llm agents fail and how they can learn from failures, 2025.

Adam Zweiger et al. Self-adapting language models. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. arXiv:2506.10943.

A Evolution Settings

A.1 Initialization and Benchmark Organization

A generic starting point. The initial package $\mathcal{M}_0$ specifies the memory interface before task-specific experience is learned. In the initial τ^2 package, $\mathcal{E}_0$ is empty, and the lists of additional invocation rules and checkers are empty. The runtime provides default working-state maintenance; the explicit status board is initially disabled. Evolution can add skills and configure state presentation, invocation, and checking through the same four-component interface. The benchmark's policy, tool schemas, and user instruction remain normal task inputs: they are not learned experience or reference answers.

How evolution is organized. All benchmarks use the Skill Memory structure and EM–WM coupling in Section 3. They differ in where the next round's evidence comes from: a designated evolution split for τ^2, the official task-family sequence for SkillFlow, and successive attempts on one task for Terminal-Bench. Table 2 makes this distinction explicit.

Table 2: **Data used to evolve and evaluate memory.** Counts describe the reference splits; model-specific evaluations use their fixed per-model partitions. SkillFlow counts task–family pairs. "Test" denotes exclusion from the corresponding run's evolution and development sets.

Setting	Evolve / dev / test	Organization
τ^2-Retail	16 / 12 / 86	Evolve using the target execution model.
τ^2-Airline	11 / 10 / 29	Evolve using the target execution model.
SkillFlow	166 pairs; 20 families	Iteratively refine memory within the official family organization.
Terminal-Bench 2.1	One task at a time	Adapt memory between attempts on the same task.

Both shared and model-specific memory are obtained from $\mathcal{M}_0$ with the same diagnose–patch–evaluate loop, using Doubao or the target model as the execution worker, respectively. On τ^2, training (evolve) tasks supply failure trajectories for diagnosis and memory updates. Dev tasks assess each candidate beyond its diagnosed failures and control admission together with its train score. In the held-out evolution protocol, test is strictly outside the loop: no test instruction, trajectory, or reward enters diagnosis, patch generation, or the acceptance gate. Memory remains fixed during test evaluation. Appendix C.2 reports outcomes on each subset. The external-method comparisons use 86 Retail and 29 Airline tasks. The external SkillFlow comparison uses a separate 114-task cohort with three trials per task. SkillFlow evaluates 166 task–family pairs. Terminal adaptation uses the task instruction, failed trajectory, and binary outcome, without hidden tests or reference outputs.

A.2 From Failure Evidence to an Admitted Memory

Figure 8 summarizes one split-based round. The Meta-Agent reads the current memory and failed evolution traces, identifies the implicated components, and produces a scoped patch. The driver validates the patch format, constructs the candidate, and reevaluates both the incumbent and candidate on all evolution tasks and the development tasks. The next round uses the admitted memory to generate new traces, so it can diagnose behavior changed by earlier repairs.

Netbest admission. Let Δn_{evolve} and Δn_{dev} denote candidate minus incumbent successful task trials on the evolution and development evaluations, respectively. The netbest configuration admits a valid candidate exactly when

$$\Delta n_{\text{evolve}} + \Delta n_{\text{dev}} > 0 \quad \text{and} \quad \Delta n_{\text{dev}} \geq 0. \tag{7}$$

These are counts over the matched task trials, not an unweighted average of split-level percentages. A gain on evolution tasks cannot offset a loss on dev. A rejected package does not replace the incumbent. The model-specific netbest runs stop at the round limit or after two consecutive gate rejections; the recursive-versus-reset control uses four rounds. Development scores are selection feedback; test scores do not decide admission.

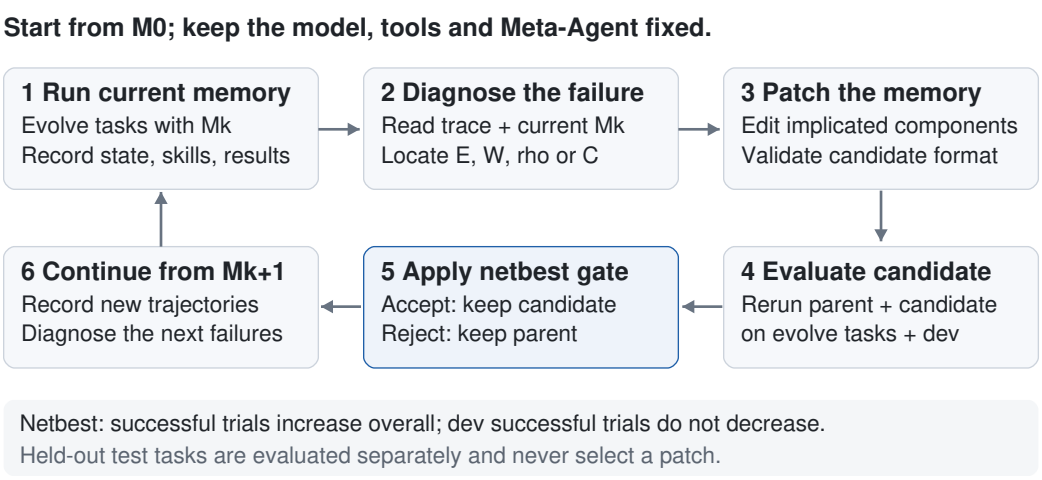


Figure 8: **An admitted memory changes the evidence available to the next round.** The split-based loop starts from $\mathcal{M}_0$; rejection retains the incumbent. Test tasks are outside the loop.

A.3 Evaluation Configuration

Default τ^2 execution uses Doubao-2.0-Pro, and the default Meta-Agent uses Doubao-2.1-Pro, both with `medium` reasoning. Sol and Opus use `xhigh`; Gemini uses the high-reasoning Flash endpoint. The default τ^2 user simulator is Doubao-2.0-Pro. The agent limit is 200 steps and ten consecutive tool errors. Model weights and environment tools remain fixed throughout evolution.

Avg@k averages binary task success across k episodes per task and then across tasks. The coupling, held-out evolution, and frontier τ^2 evaluations use $k = 4$; open-source baselines use four or eight trials per task. The Sol, Opus, and Gemini SkillFlow comparisons use $k = 1$ on the same 166 task–family pairs, with the Doubao-generated memory frozen during evaluation. Where shown, intervals use 10,000 paired task-clustered bootstrap resamples.

Controls for the main analyses. The coupling ablation holds one evolved package fixed per domain and evaluates all 114 Retail or 50 Airline tasks, four times each. "All skills in context" changes both invocation and context exposure. Tokens per success count unsuccessful episodes as well as successful ones and exclude evolution. Fault localization uses nine injected cases in each of $\mathcal{E}$, $\mathcal{W}$, and ρ, with two judgments per case: 54 judgments per evidence condition. The same Doubao-2.1-Pro judge sees outcomes, raw trajectories, or structured traces; checker faults are outside this injected set. Recursive and reset evolution use four search rounds with the same worker and gate; only the memory being patched changes. Meta-model comparisons use 86 held-out Retail tasks at $k = 4$, with each group's own remeasured $\mathcal{M}_0$.

B A Concrete Skill Memory

Figure 9 uses the illustrative exchange in Figure 2 to show what each component contains. The user wants a blue mug exchanged for the red variant. The cards show human-readable content, rather than serialization fields or a reference solution.

Working memory: what remains to be done. The original item has been identified from an order lookup, but obtaining the requested red variant remains pending. Each entry retains the goal, status, supporting observation, and any blocker. The working state keeps these facts visible without requiring the agent to reconstruct them from the entire dialogue. An associated tool identifies the intended operation; it does not by itself establish complete arguments, policy eligibility, or user confirmation.

Experiential memory: how to handle this situation. The exchange card provides a reusable procedure: identify the requested variant, establish that the operation is applicable and confirmed,

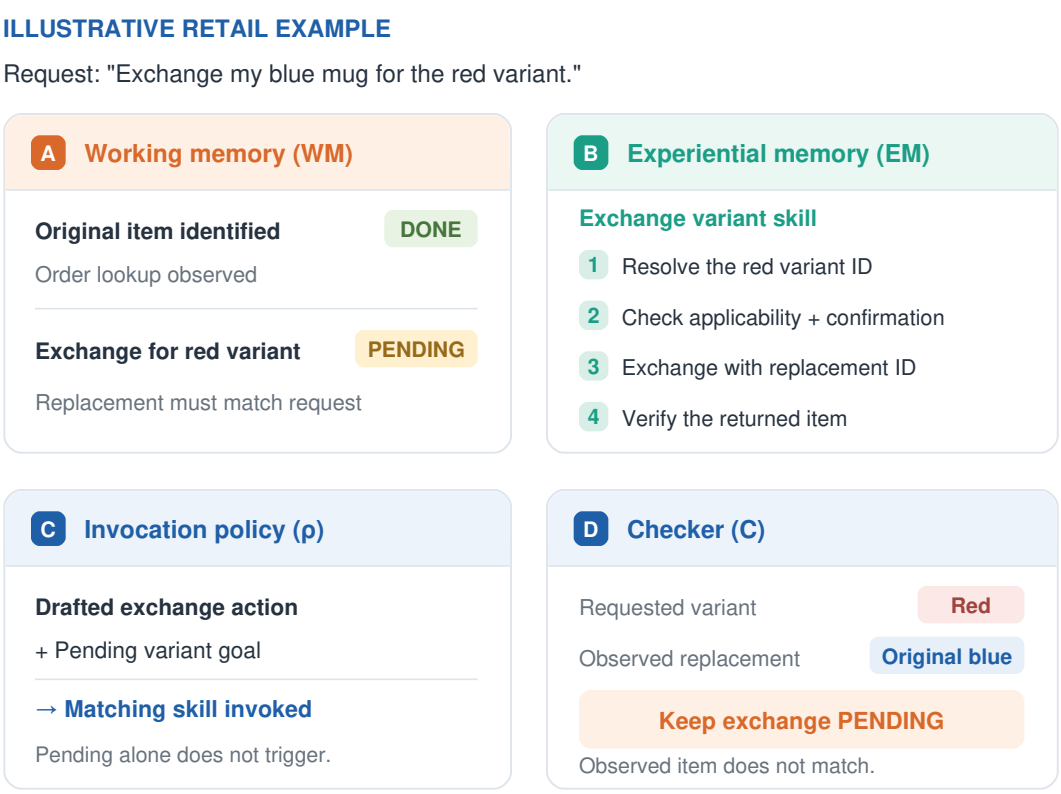


Figure 9: **A pending goal, a reusable procedure, an invocation rule, and an evidence check have distinct roles.** This illustrative Retail example displays an evolved memory, not the empty initial experience store. Names and colors are explanatory; they are not a recorded benchmark episode.

use the replacement variant's identifier, and verify the returned result. It does not store this user's order identifier. Other cards may cover returns or cancellations, but sharing a product or order with those workflows does not make them relevant to the current goal.

Invocation policy: why this card is read now. The pending exchange goal conditions the agent's next draft. When the agent drafts the exchange operation, the configured rule matches that execution event to the exchange skill's key and loads its guidance before the actual operation. A pending entry alone does not trigger every skill: the state and the relevant event jointly determine applicability. Invocation supplies guidance to the model; it does not automatically execute a tool or certify authorization.

Checker: whether the goal is actually complete. After execution, the proposed completion is compared with the available receipt and the requested outcome. An acknowledged exchange that still identifies the original blue item does not support completion of the red-variant goal. The checker rejects that transition and the fixed kernel keeps the goal pending. Programmatic receipt checks and configured semantic checks serve this evidence test; their rules are part of the memory package. The two individual controls analyzed in the main text follow these roles: pre-write invocation selects the skill, and the status board makes the working state visible.

C Additional Evaluation of Memory Evolution

Does the initial package already explain the gain? We compare the benchmark agent, the same execution stack with $\mathcal{M}_0$, and the evolved memory on Doubao-2.0-Pro and GPT-OSS-20B (Figure 10). The Doubao endpoint is the final recursive memory in Figure 7(a). These examples show how evolution builds useful behavior on top of a generic memory interface. Each model uses the same 86 held-out Retail tasks, four trials per task, and matched trial seeds, decoding settings, environment, and Doubao user simulator across its three conditions.

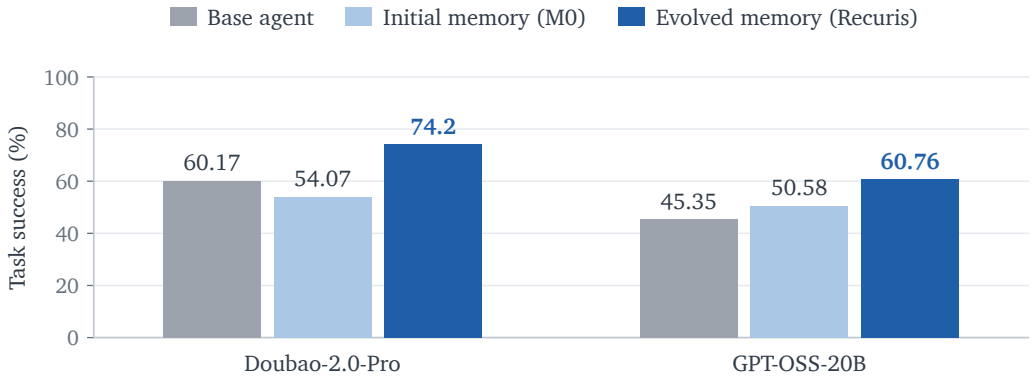


Figure 10: **Evolved memory substantially improves on both initialization and the base agent.** τ^2-Retail, 86 held-out tasks per model, avg@4. $\mathcal{M}_0$ and Recuris use the same memory-enabled stack before and after evolution.

Evolution improves success over $\mathcal{M}_0$ by 20.1 points on Doubao and 10.17 on GPT-OSS. The evolved packages also exceed the corresponding benchmark agents by 14.0 and 15.41 points. Providing an editable memory interface therefore does not explain these gains by itself: the loop must turn execution evidence into useful procedures and control rules. This is the role of evolution in Recuris, which is to specialize the initially generic memory through the failures encountered during execution.

C.1 A Matched-Attempt Check for Terminal Adaptation

To distinguish adaptation from retries (Wang et al., 2026c), we compare two Doubao-2.0-Pro conditions on the same 87 Terminal-Bench tasks, using Terminus-2. Both share the first rollout and stop at success, with at most four worker attempts. Fixed seed memory solves 51/87 tasks (58.6%); adaptation solves 53/87 (60.9%). This is a $+2.3$-point improvement at the same worker-attempt limit. Meta-Agent calls are additional computation, so it is not an equal-total-cost comparison. These first-success rates differ from the main table's avg@4 measurements over four untruncated rollouts.

C.2 Performance Beyond the Evolution Feedback

Table 3 reports task success separately on train, dev, and test. Each row uses the baseline and memory package from Table 1; the roles of train, dev, and held-out test are defined in Appendix A.

Table 3: **Task success on train, dev, and test.** Each cell gives baseline → Recuris (change in percentage points).

(a) τ^2-Retail

Model	Train	Dev	Test
Granite-4.1-3B	51.56 → 73.44 (+21.88)	43.75 → 35.42 (-8.33)	27.62 → 28.20 (+0.58)
Qwen3.5-4B	30.21 → 70.83 (+40.63)	58.33 → 66.67 (+8.33)	74.31 → 71.67 (-2.64)
Qwen3.5-9B	42.86 → 48.21 (+5.36)	64.58 → 70.83 (+6.25)	77.56 → 79.55 (+1.99)
GPT-OSS-20B	20.31 → 53.13 (+32.81)	45.83 → 60.42 (+14.58)	45.35 → 60.76 (+15.41)
Qwen3.6-27B	24.22 → 78.13 (+53.91)	54.17 → 87.50 (+33.33)	71.22 → 87.50 (+16.28)
Qwen3.6-35B	29.69 → 42.19 (+12.50)	75.00 → 72.92 (-2.08)	78.20 → 78.49 (+0.29)
GPT-5.6 Sol	45.31 → 78.13 (+32.81)	66.67 → 87.50 (+20.83)	59.59 → 74.13 (+14.53)
Claude Opus 5	62.50 → 82.81 (+20.31)	77.08 → 89.58 (+12.50)	73.55 → 88.66 (+15.12)
Gemini 3.7 Flash	60.94 → 68.75 (+7.81)	81.25 → 91.67 (+10.42)	74.71 → 78.20 (+3.49)

(b) τ^2-Airline

Model	Train	Dev	Test
Granite-4.1-3B	59.38 → 75.00 (+15.63)	57.50 → 60.00 (+2.50)	27.34 → 29.69 (+2.34)
Qwen3.5-4B	71.59 → 73.86 (+2.27)	71.25 → 78.75 (+7.50)	78.02 → 81.03 (+3.02)
Qwen3.5-9B	42.50 → 40.00 (-2.50)	75.00 → 85.42 (+10.42)	97.28 → 100.00 (+2.72)
GPT-OSS-20B	25.00 → 29.55 (+4.55)	45.00 → 51.25 (+6.25)	69.40 → 73.28 (+3.88)
Qwen3.6-27B	75.00 → 85.23 (+10.23)	68.75 → 78.75 (+10.00)	84.05 → 78.45 (-5.60)
Qwen3.6-35B	75.00 → 80.68 (+5.68)	71.25 → 72.50 (+1.25)	85.34 → 84.91 (-0.43)
GPT-5.6 Sol	68.18 → 77.27 (+9.09)	75.00 → 95.00 (+20.00)	84.48 → 86.21 (+1.72)
Claude Opus 5	90.91 → 88.64 (-2.27)	95.00 → 97.50 (+2.50)	87.07 → 88.79 (+1.72)
Gemini 3.7 Flash	86.36 → 79.55 (-6.82)	92.50 → 90.00 (-2.50)	84.48 → 85.34 (+0.86)

Repetition counts are given in Appendix A.

On Retail, Qwen3.6-27B improves from 71.22% to 87.50% on held-out tasks after updates guided by train/dev feedback. Airline also improves on held-out tasks for Granite and 9B, by 2.34 and 2.72 points. These results support accumulating procedures and control rules that remain useful beyond the failures that produced them. The effect varies across models: Retail 4B improves on train but decreases on test, showing why transfer must be evaluated separately from fitting the observed failures.

D Worked Examples of Execution and Evolution

D.1 One Within-Task Reasoning Cycle

The user wants a blue mug exchanged for its red variant. Figure 11 follows one illustrative execution with the exchange skill from Appendix B: an incorrect draft is revised before it reaches the environment, and completion is recorded only after the returned item matches the request.

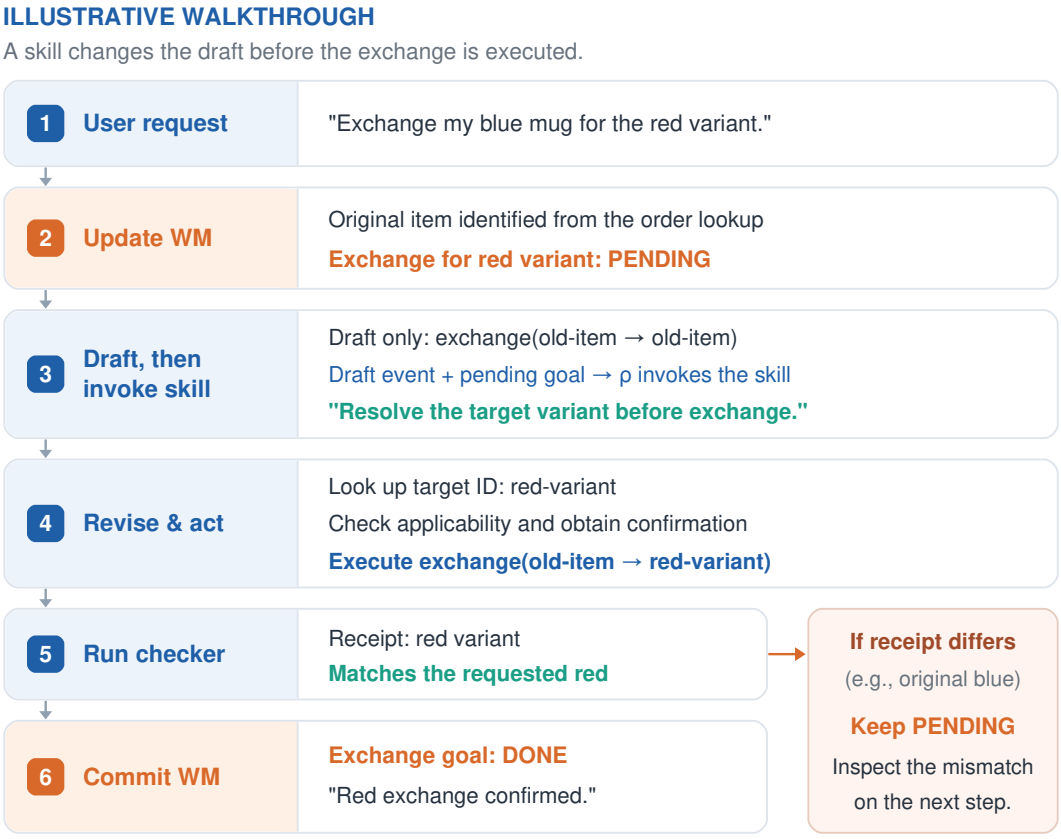


Figure 11: **The current goal determines which experience is invoked; the actual result determines whether that goal is complete.** An illustrative Retail exchange, with schematic item identifiers. The draft in step 3 is not executed. Steps 4–6 show the corrected action and verified update; the side branch shows how a mismatching receipt keeps the goal pending.

The key transition occurs between drafting and acting. WM makes the unresolved variant requirement visible; ρ uses the drafted operation to invoke the exchange procedure; the procedure changes the tool arguments. The checker then compares the result with the original requirement. A skill can therefore change the action without granting itself permission to declare success. The trace records each of these decisions for later diagnosis.

D.2 One Across-Task Memory Update

Figure 12 follows the first accepted update of a Qwen3.6-27B Retail run. The saved diagnosis identifies a same-item exchange in task 18, trial 1. The new exchange card requires a replacement with different options, a summary of the selected items, explicit confirmation, and the actual exchange call. The candidate also configures request tracking, intent-triggered invocation, and checkers, so the new procedure can affect execution at the relevant step.

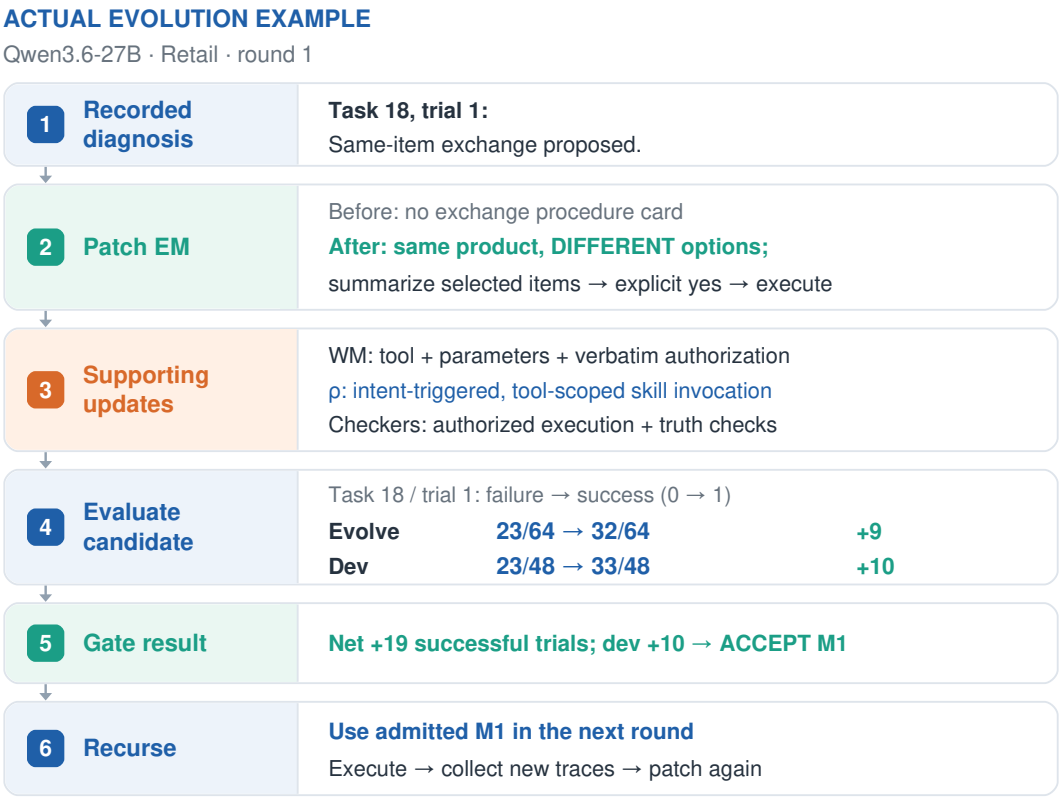


Figure 12: **A concrete execution failure becomes a reusable procedure, then a validated memory update.** Saved Qwen3.6-27B Retail diagnosis, package edits, and round-1 gate results. One exchange card is highlighted; the gate evaluates the complete candidate, including its other card and control changes. Counts are successful trials, with four trials on each of 16 evolve and 12 dev tasks.

The identified task–trial changes from failure to success under the candidate. Across the full gate evaluation, the complete candidate adds nine successful evolution trials and ten dev successes. Netbest therefore retains it as $\mathcal{M}_1$, and subsequent rounds generate and diagnose traces under this new memory. The failure contributes more than a reminder to be careful: it changes what the exchange procedure requires and when that procedure is invoked. Admission then tests the combined change before it becomes the starting point for further evolution.