

EVO-HARNESS: Context-to-Harness Skill Compilation for Self-Evolving Agents

Tianxin Wei¹, Zhan Shi², Minhua Lin³, Bing He², Zewen Liu⁴, Yisi Sang²,
Yuanchen Bei¹, Xuying Ning¹, Jiaru Zou¹, Ting-Wei Li¹, Xiao Lin¹, Yanjun Zhao¹,
Chi Wang⁵, Benoit Dumoulin², Dakuo Wang⁶, Jingrui He¹, Hanqing Lu²

¹University of Illinois Urbana-Champaign ²Amazon ³The Pennsylvania State University
⁴Emory University ⁵AG2 AI ⁶Northeastern University

Abstract

Learning from experience is critical for developing capable, self-improving large language model (LLM) agents. Existing methods typically extract knowledge from accumulated trajectories via reflection, memory, rules, or skills. However, agents in realistic environments continuously encounter novel tasks, often offering only a one-shot opportunity to improve. These executions yield rich but highly noisy contexts, entangling broadly useful lessons with task-specific artifacts. Critically, prior works rarely validate their effectiveness on complex real-world tasks or isolate the underlying drivers of improvement. To address these gaps, we study *online harness learning*, where a frozen agent improves by continually updating a structured harness across sequential tasks. This setup enables a systematic study of key self-improvement factors through *Evo-Harness*. At its core, *context-to-harness skill compilation* distills noisy, single-shot executions into reusable skill harnesses for cross-domain and topic-level adaptation. To demonstrate the efficacy of one-shot skill compilation, we evaluate across five realistic benchmarks (TerminalBench2, SWE-bench, CL-Bench, τ -bench, WebArena-Infinity). Our extensive analysis demonstrates the effectiveness of EVO-HARNESS and provides a principled understanding of how LLM agents can effectively learn on the fly. Our code is available at [Link](#).

1 Introduction

Large language model (LLM) agents (Achiam et al., 2023; Team et al., 2023; Liu et al., 2024) have shown strong performance in complex task-solving settings, such as web interaction (Zhou et al., 2024), software engineering (Jimenez et al., 2024), tool use (Qin et al., 2024), and long-horizon reasoning (Erdogan et al., 2025; Sun et al., 2025). These settings require agents to plan, act, interpret observations, handle feedback, and recover from errors. However, when an agent fails on a task, the

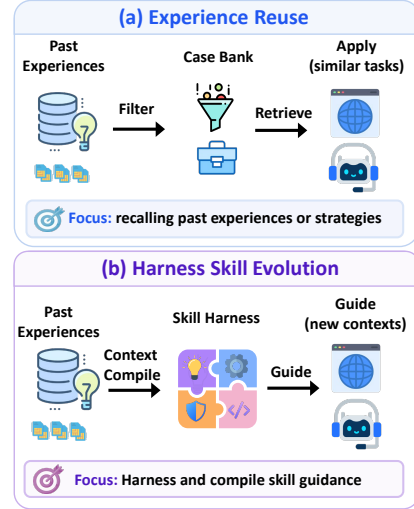


Figure 1: Comparison of experience reuse (retrieving past cases) and harness skill evolution (compiling experiences into reusable guidance).

failure often remains an isolated event rather than a source of future improvement. Without converting failed executions into reusable lessons, agents continue to make similar mistakes in later tasks.

Many experience-based methods (Shinn et al., 2023; Zhao et al., 2024) address this issue by learning from accumulated trajectories offline (Lee et al., 2026; Ni et al., 2026), utilizing media such as reflection (Shinn et al., 2023; Qiu et al., 2026), memory (Zhong et al., 2024; Chhikara et al., 2025; Xu et al., 2025), and skills (Zhang et al., 2026a; Xia et al., 2026; Jiang et al., 2026; Yang et al., 2026). However, in realistic settings, agents continuously encounter novel tasks and contexts, and each completed execution typically offers only a one-shot opportunity to improve future behavior. Furthermore, real executions produce rich but highly noisy contexts, entangling useful lessons with task-specific details, partial failures, and incidental tool traces.

Critically, prior methods often fail to rigorously validate their effectiveness under these realistic constraints, nor do they systematically isolate the underlying drivers of agent improvement. To navigate

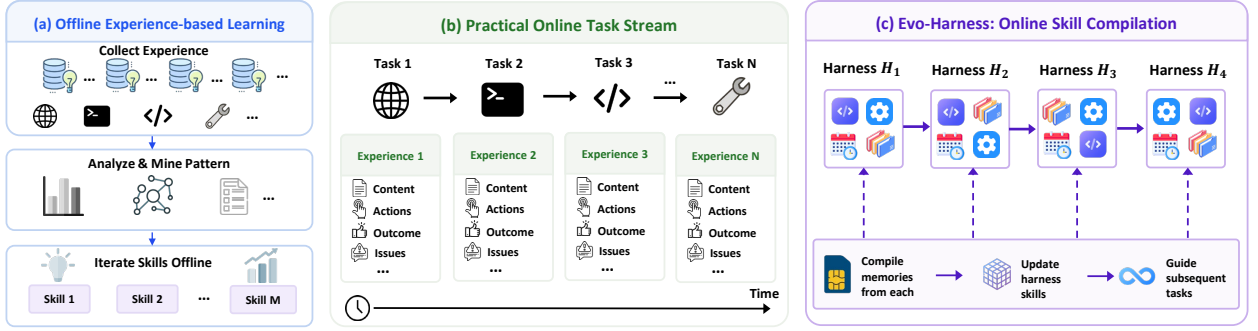


Figure 2: Illustration of the problem setting. (a) Prior methods collect executions and mine reusable patterns offline. (b) In practical online streams, agents encounter tasks sequentially, yielding rich but one-shot execution experiences. (c) EVO-HARNESS compiles these online experiences into evolving skill harnesses for future reuse.

such a one-shot, complex learning environment, relying on the simple retrieval of past experiences falls short. Instead, agents need to actively compile raw execution contexts into reusable skill guidance for novel tasks, as contrasted in Figure 1.

Building on this view, we study *online harness learning*, where a frozen agent improves by incrementally updating a structured, external harness across sequential tasks. To cope with the continuous influx of novel tasks, this harness acts as a self-evolving learning medium that progressively distills executions into actionable guidance for the future. An overview of this practical setting, compared to standard offline learning, is illustrated in Figure 2.

To operationalize this formulation and enable a systematic study of continuous self-improvement, we introduce *Evo-Harness*. We instantiate the harness within this framework using skills. Skills act as a natural medium for realistic task executions because they can integrate diverse learning signals, such as failures, feedback, constraints, and operational procedures, into structured knowledge that shapes future planning and recovery. At the core of *Evo-Harness* is *context-to-harness skill compilation*, which converts individual execution contexts into reusable skill harness updates. Specifically, our compilation process uses a *solver* to reflect on contexts and propose candidate skills, and an *evolver* to update the existing harness. *General* skills capture transferable cross-task patterns, while *topic* skills capture localized operational knowledge. This dual design preserves critical learning signals while filtering out trajectory-level noise.

We evaluate Evo-Harness across five realistic benchmarks (TerminalBench2, SWE-bench, CL-Bench, τ -bench, and WebArena-Infinity) under a sequential continuous learning setup. Beyond demonstrating performance gains, our primary fo-

cus is systematically investigating the core drivers of self-improvement. By analyzing how evolver designs, feedback types, and transfer settings dictate performance, our study treats the skill harness as an interpretable object for understanding how frozen LLM agents learn on the fly.

Our contributions are summarized as follows:

- **Online harness learning.** We study a practical setting where frozen agents improve over sequential task streams by incrementally updating a reusable external harness instead of model parameters.
- **Context-to-harness skill compilation.** We introduce a mechanism that compiles single-shot execution contexts into actionable general and topic-level skill updates.
- **Systematic analysis of harness evolution.** We isolate how evolver designs, feedback types, and transfer settings impact continuous self-improvement. Evaluated across five complex benchmarks, EVO-HARNESS consistently outperforms existing experience-based methods.

2 Related Work

In this section, we review existing works on self-evolving LLM agents and external memory or skill harnesses for agent experience.

Evolving and self-improving LLM agents. A growing line of work explores how LLM agents self-improve through interaction and feedback. Beyond local reflection methods that iteratively revise outputs based on scalar or verbal feedback (Shinn et al., 2023; Madaan et al., 2023), experience-based approaches attempt to extract reusable knowledge offline. For instance, some works mine accumulated trajectory pools to synthesize natural language insights (Zhao et al., 2024), interaction

rules (Chen et al., 2024), or transferable directories (Ni et al., 2026). Similarly, a parallel track focuses on passively evolving higher-level agent artifacts, such as prompts (Suzgun et al., 2025; Lou et al., 2026), reasoning banks (Ouyang et al., 2025), or memory structures (Agrawal et al., 2025; Lee et al., 2026; Zhang et al., 2025; Wei et al., 2025; Lin et al., 2026a). To provide more operational guidance, recent methods have shifted toward skill-centered evolution. This includes building executable code skills (Wang et al., 2023), optimizing skill libraries via policy learning (Xia et al., 2026; Zhang et al., 2026a; Ouyang et al., 2026), or curating skill knowledge bases and representations (Yang et al., 2026; Qiu et al., 2026; Jiang et al., 2026; Wang et al., 2026). While closely related, these existing frameworks largely treat learned artifacts as passive repositories for free-form retrieval. Critically, prior methods often fail to rigorously validate their effectiveness under realistic online constraints, nor do they systematically isolate the underlying drivers of agent improvement. Our work addresses this gap by compiling single-shot experiences into an active skill harness, providing a structural scaffold to systematically investigate how LLM agents learn on the fly.

Memory and skill harnesses for agent experience. Memory provides a fundamental external medium for preserving agent experience. Existing memory systems maintain long-term observation histories (Zhong et al., 2024; Yu et al., 2025), organize memories through adaptive structures (Xu et al., 2025), and explore methods for updating and retrieving evidence across modalities (Huang et al., 2026; Bei et al., 2026; Liu et al., 2026a). However, these approaches largely treat memory as a passive information store. Skills differ fundamentally because they encode operational knowledge. They shift from merely recording what happened to specifying exactly how an agent should act through manuals (Chen et al., 2024), verify operations via organized directories (Ni et al., 2026), or recover from failures using skill repositories (Ouyang et al., 2026; Wang et al., 2026; Zhang et al., 2026b). This procedural view elevates skills into an active harness. Rather than merely supplying historical context, a harness acts as an external scaffold that structurally shapes future planning, taking forms such as pattern repositories (Ning et al., 2026; Qiu et al., 2026), reasoning banks (Ouyang et al., 2025), and dynamic guidance files (Suzgun et al., 2025; Lee

et al., 2026; Liu et al., 2026b; Lin et al., 2026b). Building on this distinction, our work conceptualizes the skill harness not merely as a performance-enhancing artifact, but as an interpretable medium to systematically study continuous online adaptation. To navigate the high noise of real-world executions, EVO-HARNESS compiles single-shot reflections into a structured harness of general and topic skills. While this separation efficiently filters out task-specific artifacts, its primary purpose in our work is to provide a transparent scaffold. This explicitly allows us to systematically isolate and evaluate the core drivers, such as evolver designs and feedback types, that dictate how frozen LLM agents self-improve on the fly.

3 EVO-HARNESS: Context-to-Harness Skill Compilation

We present EVO-HARNESS, a framework for online harness learning. The goal is to convert one-shot execution contexts into a reusable harness that guides future task solving. Unlike experience retrieval, which recalls past cases, EVO-HARNESS compiles noisy execution signals into structured guidance that can shape later planning, acting, verification, and recovery.

Figure 3 gives an overview. EVO-HARNESS maintains a current harness that guides the solver before execution, shaping how it plans, acts, verifies, and recovers. After each task batch, execution contexts are reflected into candidate memories and compiled back into the harness as cross-task patterns and task-type procedures, allowing the updated harness to guide later executions.

3.1 Online Harness Learning Setup

Let $\{B_1, \dots, B_K\}$ denote a stream of task batches, where $B_i = \{x_{i,1}, \dots, x_{i,m_i}\}$. Before batch B_i , the agent has a current harness $\mathcal{H}_i$, a structured set of reusable guidance entries:

$$\mathcal{H}_i = \{h_i^1, \dots, h_i^{n_i}\}. \quad (1)$$

The harness is external to the frozen solver and serves as a control scaffold for later executions. Unlike raw trajectory storage, it maintains compact guidance distilled from prior executions and affects future behavior through selection and injection into the solver context.

For each task $x_{i,j}$, the agent selects a compact set of harness entries:

$$\mathcal{S}_{i,j} = \text{Select}(x_{i,j}, \mathcal{H}_i; b), \quad |\mathcal{S}_{i,j}| \leq b, \quad (2)$$

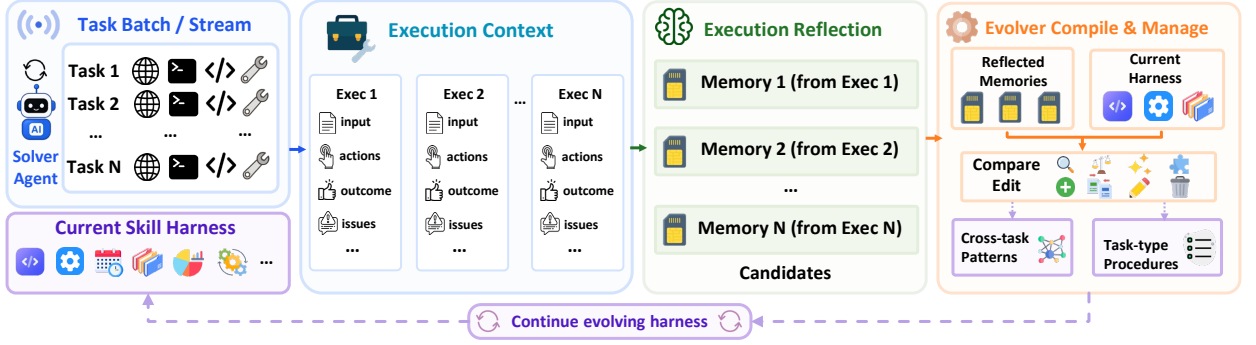


Figure 3: Overview of EVO-HARNES. The solver uses the current harness to solve a task batch or stream. Each execution produces a context, including input, action trajectory, outcome, and feedback. Execution reflection converts these contexts into candidate memories. The evolver compares the candidate memories with the current harness and updates the harness with cross-task patterns and task-type procedures for later batches.

where b is the injection budget. The selected guidance is injected into the task:

$$\tilde{x}_{i,j} = \text{Inject}(x_{i,j}, \mathcal{S}_{i,j}). \quad (3)$$

A frozen solver $\mathcal{A}$ executes the task:

$$(\tau_{i,j}, y_{i,j}, f_{i,j}) = \mathcal{A}(\tilde{x}_{i,j}), \quad (4)$$

where $\tau_{i,j}$ is the action trajectory, $y_{i,j}$ is the outcome, and $f_{i,j}$ is available feedback, such as verifier results, unit-test outputs, tool diagnostics, or judge feedback. The execution context is:

$$c_{i,j} = (x_{i,j}, \tau_{i,j}, y_{i,j}, f_{i,j}). \quad (5)$$

Online harness learning updates $\mathcal{H}_i$ into $\mathcal{H}_{i+1}$ from these execution contexts, while keeping the solver parameters fixed.

3.2 Context-to-Harness Compilation

EVO-HARNES uses two complementary stages: *execution reflection* and *harness evolution*. Reflection extracts learning signals from each execution. Evolution integrates those signals into the harness.

Execution reflection. For each completed execution context $c_{i,j}$, the solver produces a candidate memory only when the execution fails or receives negative feedback:

$$r_{i,j} = \text{Reflect}(c_{i,j}). \quad (6)$$

Each candidate memory is represented as:

$$r = (\text{lesson}, \text{trigger}, \text{evidence}, \text{scope_hint}). \quad (7)$$

Reflection uses the full task-side context available immediately after execution, including the instruction, trajectory, observations, tool outputs,

outcome, and feedback. We focus reflection on failures because they expose the solver’s current boundary: incorrect assumptions, missing constraints, ineffective tool use, weak verification, and failed recovery. This makes the resulting candidate memories more targeted for harness improvement, while avoiding unnecessary updates from successful executions that may contain task-specific details. The reflection stage does not compare against the current harness, so its output is not a final harness update, but a failure-grounded candidate signal extracted from the completed execution.

Harness evolution. After batch B_i , the candidate memories form:

$$\mathcal{R}_i = \{r_{i,j}\}_{j=1}^{m_i}. \quad (8)$$

The evolver receives the candidate memories and the current harness:

$$\mathcal{O}_i = \text{Evolver}(\mathcal{H}_i, \mathcal{R}_i), \quad (9)$$

where $\mathcal{O}_i$ is a set of harness edits. Conceptually, the evolver applies:

$$o = \pi_{\text{evolve}}(r, \mathcal{H}_i, \mathcal{R}_i), \quad (10)$$

where $o \in \{\text{ADD}, \text{MERGE}, \text{REVISE}, \text{SKIP}\}$. The evolver filters noisy or redundant memories, merges compatible signals, revises existing guidance, and promotes reusable lessons when appropriate. The updated harness is:

$$\mathcal{H}_{i+1} = \text{ApplyEdits}(\mathcal{H}_i, \mathcal{O}_i). \quad (11)$$

This makes the harness more than a memory store: candidate memories are not appended directly, but compiled into guidance that can affect future behavior.

Algorithm 1 EVO-HARNESS: Context-to-Harness Skill Compilation

```
1: Input: task batches  $\{B_1, \dots, B_K\}$ , frozen solver  $\mathcal{A}$ ,  
   initial harness  $\mathcal{H}_1$   
2: for  $i = 1, \dots, K$  do  
3:    $\mathcal{R}_i \leftarrow \emptyset$   
4:   for each task  $x_{i,j} \in B_i$  do  
5:      $\mathcal{S}_{i,j} \leftarrow \text{Select}(x_{i,j}, \mathcal{H}_i; b)$   
6:      $\tilde{x}_{i,j} \leftarrow \text{Inject}(x_{i,j}, \mathcal{S}_{i,j})$   
7:      $(\tau_{i,j}, y_{i,j}, f_{i,j}) \leftarrow \mathcal{A}(\tilde{x}_{i,j})$   
8:      $c_{i,j} \leftarrow (x_{i,j}, \tau_{i,j}, y_{i,j}, f_{i,j})$   
9:      $r_{i,j} \leftarrow \text{Reflect}(c_{i,j})$   
10:     $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{r_{i,j}\}$   
11:  end for  
12:   $\mathcal{O}_i^{\text{type}} \leftarrow \text{CompileTaskType}(\mathcal{H}_i, \mathcal{R}_i)$   
13:   $\mathcal{O}_i^{\text{cross}} \leftarrow \text{CompileCrossTask}(\mathcal{H}_i, \mathcal{R}_i)$   
14:   $\mathcal{O}_i \leftarrow \mathcal{O}_i^{\text{type}} \cup \mathcal{O}_i^{\text{cross}}$   
15:   $\mathcal{H}_{i+1} \leftarrow \text{ApplyEdits}(\mathcal{H}_i, \mathcal{O}_i)$   
16: end for  
17: Output: evolved harness  $\mathcal{H}_{K+1}$  and task results
```

3.3 Harness Levels

The evolved harness contains two levels of reusable guidance. *Cross-task patterns* are compiled by comparing memories across tasks and identifying shared operational structure. *Task-type procedures* preserve more localized guidance for recurring task formats, interfaces, or domains. These levels are not predefined labels for individual executions. They emerge from how the evolver filters, compares, and integrates candidate memories into the harness.

3.4 Algorithm and Modularity

Algorithm 1 summarizes EVO-HARNESS. We use batch-level updates so the evolver can both compile localized task-type procedures and compare memories across tasks to extract cross-task patterns. The same formulation reduces to task-type updating when $|B_i| = 1$.

Here, `CompileTaskType` preserves localized procedures, while `CompileCrossTask` extracts shared patterns across executions. The modular design is intentional: by decoupling the solver, evolver, feedback source, and initial harness state, EVO-HARNESS provides a controlled framework for systematically studying what drives online harness improvement.

4 Experiments

We conduct experiments to evaluate EVO-HARNESS and to understand when online harness evolution improves LLM agents:

- **RQ1: Overall effectiveness.** Does EVO-HARNESS improve frozen agents across diverse

benchmarks, and which task categories benefit most?

- **RQ2: Harness content and components.** What types of guidance are produced by the evolved harness, and how do different harness components contribute?
- **RQ3: Online updating and transfer.** How do evolved harnesses behave under train-split transfer, online updating, and solver-evolver pairing?
- **RQ4: Feedback grounding.** How does the source and granularity of feedback affect harness evolution?

4.1 Setup

Experimental setup. We evaluate EVO-HARNESS on five benchmarks spanning diverse agent capabilities: WebArena-Infinity (Zhou, 2026) for web navigation, TerminalBench2 (Merrill et al., 2026) for command-line tasks, SWE-bench Lite (Jimenez et al., 2024) for software engineering, CL-Bench (Dou et al., 2026) for adaptive reasoning, and TAU-Bench (Yao et al., 2024) for tool-use tasks. We use Claude Opus 4.6 (Anthropic, 2026a) as the default solver and evolver, and additionally evaluate with Claude Opus 4.7 (Anthropic, 2026b), Claude Opus 4.5 (Anthropic, 2025), Kimi-K2.5 (Team et al., 2026), and GPT-OSS (Agarwal et al., 2025) when studying model ablations. We report success rates in percent, where higher is better. Further benchmark and implementation details are provided in Appendix A and F.

Baselines. We compare with representative experience-, memory-, and skill-based adaptation methods under the same task split and evaluation protocol. AWM (Wang et al., 2024) reuses agent working memory, Dynamic Cheatsheet (Suzgun et al., 2025) maintains an adaptive external cheatsheet, Evo-Memory (Wei et al., 2025) evolves reusable memories from executions, ACE (Zhang et al., 2025) optimizes agent context, and XSkill (Jiang et al., 2026) separates experience and skills for continual reuse. DC-Cu and DC-RS are two Dynamic Cheatsheet variants.

4.2 Overall Effectiveness (RQ1)

Table 1 reports the main results with Claude Opus 4.6 as the solver. **EVO-HARNESS improves over both No-Evolve and prior experience-based baselines across all five benchmarks.** The gains are especially large on TerminalBench-2, where agents need to inspect files, execute commands, in-

Table 1: Main results using Claude Opus 4.6 as the solver. Scores denote success rates (%), and higher values indicate better performance. EVO-HARNESS achieves the best results across all five benchmarks.

Method	CL-Bench	TerminalBench-2	SWE-bench Lite	τ -bench	WebArena-Infinity
No Evolve	29.54	62.92	63.67	72.73	72.50
AWM	28.91	62.92	62.67	70.91	70.00
DC-Cu	29.23	60.67	63.00	72.12	71.25
DC-RS	29.07	61.80	61.67	70.91	70.00
Evo-Memory	29.38	64.04	64.00	72.12	72.50
ACE	29.70	61.80	63.67	72.73	71.25
XSkill	31.44	66.29	64.67	73.94	73.75
EVO-HARNESS	34.02	73.03	67.00	76.97	76.25

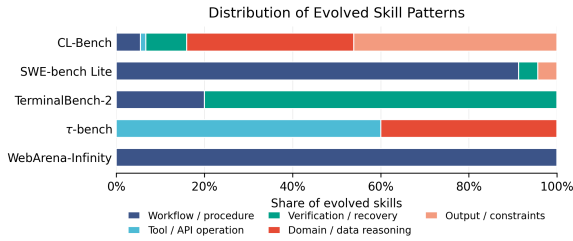


Figure 4: Distribution of evolved skill patterns across benchmarks. The learned skills exhibit benchmark-specific characteristics, ranging from workflow and tool-use procedures to verification, domain reasoning, and output-constraint handling.

interpret errors, and recover from failed operations. This suggests that harness evolution is particularly useful when tasks expose reusable procedural structure.

Compared with the strongest baseline, XSkill, EVO-HARNESS still improves on every benchmark. **External experience reuse is not automatically beneficial.** Several baselines underperform No-Evolve on some benchmarks, suggesting that retrieved or evolved guidance can be noisy, overly specific, or mismatched to the solver. This supports our motivation for studying not only whether self-evolution helps, but also what factors make online harness updates effective.

4.3 Category and Model Generality (RQ1)

Table 2 provides a representative category-level view on CL-Bench across five solver models. Since CL-Bench covers diverse reasoning and execution categories, it allows us to examine where harness evolution is most effective and how the effect varies across solver capabilities. **EVO-HARNESS improves the overall score for every solver model, with larger gains on stronger Claude models than on open-weight alternatives.** Opus 4.7, Opus 4.6, and Opus 4.5 obtain gains of +3.7, +4.5,

and +3.8, respectively, while Kimi-K2.5 and GPT-OSS improve by +1.1 and +0.8. This suggests that stronger solvers may be better able to interpret, follow, and benefit from evolved harness guidance.

Across categories, the most consistent improvements appear on Procedural Task Execution (PTE), where every model improves and Opus 4.6 and Opus 4.5 each gain +9.8 points. **Harness evolution is most reliable when tasks expose reusable operational structure.** In contrast, Empirical Discovery & Simulation (EDS) shows less stable gains, with drops for Opus 4.6 and GPT-OSS. This suggests that more exploratory tasks may be more sensitive to overly specific guidance.

4.4 Harness Content and Components (RQ2)

Figure 4 analyzes the content of the evolved harness across benchmarks. **The evolved harness reflects benchmark-specific operational demands rather than accumulating generic guidance.** WebArena-Infinity mainly yields workflow and procedure skills, TerminalBench-2 emphasizes verification and recovery, τ -bench mixes tool/API operation with domain reasoning, and CL-Bench produces a broader distribution including domain reasoning and output-constraint handling. These patterns suggest that context-to-harness compilation adapts the harness to each environment’s operational structure and failure modes.

Table 3 studies which parts of the harness update process contribute to performance. We focus this and the following diagnostic analyses mainly on CL-Bench and SWE-bench Lite because they stress different forms of harness guidance: CL-Bench contains heterogeneous adaptive reasoning tasks where localized task-type guidance can matter, while SWE-bench Lite emphasizes repository-level debugging and verification procedures where broader reusable guidance is expected to help. No

Table 2: CL-Bench category results across different models: pass rate (%) across five models. **Bold** = best per category-model pair. Δ shows absolute improvement of EVO-HARNESS over the No-Evolve baseline. DKR = Domain Knowledge Reasoning, EDS = Empirical Discovery & Simulation, PTE = Procedural Task Execution, RSA = Rule System Application.

Model	Method	DKR		EDS		PTE		RSA		All	
		Rate	Δ	Rate	Δ	Rate	Δ	Rate	Δ	Rate	Δ
Opus 4.7	No-Evolve	32.4	—	19.1	—	35.5	—	29.0	—	30.8	—
	EVO-HARNESS	34.1	(+1.7)	19.6	(+0.5)	42.7	(+7.2)	32.7	(+3.7)	34.5	(+3.7)
Opus 4.6	No-Evolve	31.4	—	20.6	—	34.0	—	26.9	—	29.5	—
	EVO-HARNESS	35.6	(+4.2)	18.1	(-2.5)	43.8	(+9.8)	30.4	(+3.5)	34.0	(+4.5)
Opus 4.5	No-Evolve	31.4	—	18.1	—	27.8	—	25.3	—	27.3	—
	EVO-HARNESS	32.6	(+1.2)	19.1	(+1.0)	37.6	(+9.8)	28.3	(+3.0)	31.1	(+3.8)
Kimi-K2.5	No-Evolve	25.6	—	12.1	—	25.1	—	21.6	—	22.9	—
	EVO-HARNESS	25.9	(+0.3)	13.1	(+1.0)	29.3	(+4.2)	20.8	(-0.7)	23.9	(+1.1)
GPT-OSS	No-Evolve	17.6	—	11.6	—	17.6	—	16.4	—	16.6	—
	EVO-HARNESS	17.9	(+0.3)	10.6	(-1.0)	18.5	(+0.8)	18.6	(+2.1)	17.5	(+0.8)

Table 3: Ablation study on CL-Bench and SWE-bench Lite using Claude Opus 4.6 as the solver. Scores denote pass rates (%).

Method	CL-Bench	SWE-bench Lite
No Evolve	29.54	63.67
No Propose	33.28	65.33
General Only	30.28	66.67
Topic Only	33.70	64.33
EVO-HARNESS	34.02	67.00

PROPOSE removes solver-side proposal of candidate memories, GENERAL ONLY keeps only cross-task guidance, and TOPIC ONLY keeps only localized task-type guidance.

Both broad cross-task guidance and localized task-type guidance are useful, but their importance depends on the task setting. The full EVO-HARNESS setting performs best on both benchmarks. On CL-Bench, Topic Only reaches 33.70, close to the full result of 34.02, while General Only drops to 30.28, suggesting that localized task-type guidance is more useful for its heterogeneous reasoning tasks. On SWE-bench Lite, General Only reaches 66.67, close to the full result of 67.00, while Topic Only reaches 64.33, indicating that broad debugging, verification, and repository-inspection procedures matter more. No Propose improves over No Evolve but remains below the full method, suggesting that proposal and harness-level compilation are complementary.

4.5 Transfer and Adaptability (RQ3)

Figure 5 compares three skill-update settings. NO-EVOLVE uses no harness update. TRAIN-SPLIT

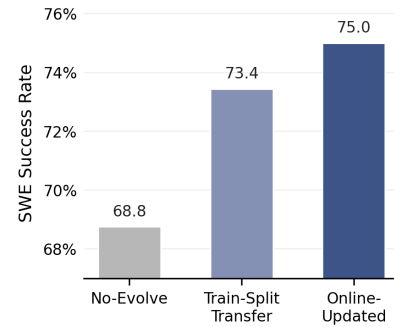


Figure 5: SWE-bench Lite success rates under different evolution settings. NO-EVOLVE uses no harness updates; TRAIN-SPLIT TRANSFER evaluates skills evolved by Claude Sonnet on the training split; and ONLINE-UPDATED updates skills during the task stream.

TRANSFER uses skills evolved by Claude Sonnet 4.5 on the training split and evaluates them with an Opus 4.7 solver on the test split. ONLINE-UPDATED updates the harness directly with the same models during the test task stream. **Skills learned on a training split can transfer across both tasks and models, while online updating performs best.** No-Evolve reaches 68.8, Train-Split Transfer reaches 73.4, and Online-Updated reaches 75.0. The gain from Train-Split Transfer shows that skills evolved by a smaller or different model can still provide useful guidance for a stronger solver on unseen tasks. Online-Updated further improves over Train-Split Transfer by +1.6 points, suggesting that in-situ harness updates better adapt to the local distribution and failure modes of the test stream.

Figure 6 examines how solver-evolver pairing affects harness evolution. We compare two solver

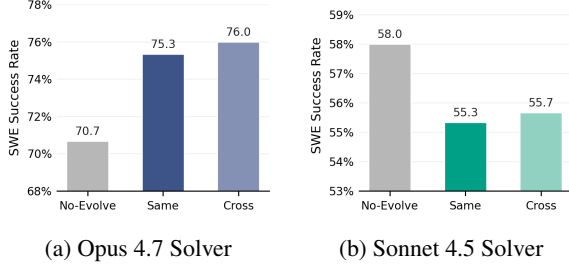


Figure 6: Effect of solver-evolver pairing on SWE-bench Lite. SAME denotes using the same model for solving and skill evolution, while CROSS denotes using different models for the solver and evolver.

settings: an Opus 4.7 solver and a Sonnet 4.5 solver. In each setting, SAME uses the same model as both solver and evolver, while CROSS uses the other model as the evolver. Specifically, for the Opus solver, SAME uses Opus to evolve the harness and CROSS uses Sonnet to evolve the harness; for the Sonnet solver, SAME uses Sonnet as the evolver and CROSS uses Opus as the evolver. **Using the same model for solving and evolution is not always optimal; cross-model evolution can help, but only when the solver is capable of using the evolved guidance.** For the Opus solver, Cross reaches 76.0, slightly higher than Same at 75.3 and clearly above No-Evolve at 70.7. This suggests that a different evolver can produce guidance that is still useful, or even more useful, when the downstream solver has sufficient capability to interpret and apply it. For the Sonnet solver, however, Same and Cross reach 55.3 and 55.7, both below No-Evolve at 58.0. Thus, Sonnet-evolved skills can help Opus, but do not necessarily help Sonnet itself. This contrast suggests that harness transfer depends not only on the skill artifact but also on the base solver’s capability: if the solver cannot reliably follow, contextualize, or adapt the guidance, evolution may fail to improve performance.

4.6 Feedback Grounding (RQ4)

Table 4 evaluates how feedback signals affect harness evolution on CL-Bench and SWE-bench Lite. We compare three feedback settings. SELF-GENERATED asks the LLM itself to judge whether the execution succeeded and what should be learned. MINIMAL provides only the environment-level success or failure signal, such as whether a command, verifier, or test passed. STANDARD provides grounded diagnostic feedback, including failure messages, error traces, verifier outputs, or test failures when available.

Table 4: Impact of feedback level on CL-Bench and SWE-bench Lite using Claude Opus 4.6 as the solver. Scores denote pass rates (%).

Feedback Level	CL-Bench	SWE-bench Lite
No Evolve	29.54	63.67
Self-Generated	27.96	61.67
Minimal	29.86	67.33
Standard	34.02	67.00

Grounded feedback is important for reliable harness evolution. Self-Generated feedback underperforms No-Evolve on both benchmarks, decreasing CL-Bench from 29.54 to 27.96 and SWE-bench Lite from 63.67 to 61.67. This suggests that LLM-generated self-judgment can introduce misleading updates when it is not grounded in external execution evidence. In contrast, Minimal and Standard feedback both use environment-provided signals and are more reliable overall.

Standard feedback is generally preferable, but the useful granularity of feedback depends on the benchmark. On CL-Bench, Standard feedback performs best, reaching 34.02, while Minimal feedback reaches only 29.86. This suggests that richer diagnostics are useful for adaptive reasoning tasks, where understanding the reason for failure can help compile better harness guidance. On SWE-bench Lite, however, Minimal feedback reaches 67.33, slightly above Standard feedback at 67.00. One possible explanation is that detailed error messages can sometimes make the evolved harness overly tied to task-specific failures, while sparse pass/fail signals encourage more generalizable guidance.

5 Conclusion

We formulated *online harness learning* as a setting for studying how frozen LLM agents can improve from sequential, one-shot executions. Through EVO-HARNESS, we instantiated this setting with context-to-harness skill compilation, which turns noisy execution contexts into a reusable external harness for future task solving. Beyond proposing a pipeline, our goal is to use the harness as an analytical lens for understanding online self-improvement. This perspective moves self-evolving agent research beyond aggregate success rates toward a more principled study of how agents produce, organize, transfer, and apply reusable experience in realistic task streams.

Limitations

Our study focuses on single-agent LLM systems in text-based, tool-use, web, command-line, and software-engineering environments. We do not evaluate embodied agents or multi-agent systems, where the harness may involve additional forms of interaction, coordination, and environmental feedback. We also instantiate the harness with natural-language guidance; other harness formats, such as executable code skills or structured programs, are not covered in this study. We also focus on scalable automatic feedback and do not evaluate expert human-edited feedback. Human-in-the-loop harness refinement and estimating the corresponding human-feedback ceiling remain important directions for future study.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.
- Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziemis, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*.
- Anthropic. 2025. Claude opus 4.5 system card. <https://www-cdn.anthropic.com/bf10f64990cfda0ba858290be7b8cc6317685f47.pdf>.
- Anthropic. 2026a. Claude opus 4.6 system card. <https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf>.
- Anthropic. 2026b. Claude opus 4.7 system card. <https://www.anthropic.com/claude-opus-4-7-system-card>.
- Yuanchen Bei, Tianxin Wei, Xuying Ning, Yanjun Zhao, Zhining Liu, Xiao Lin, Yada Zhu, Hendrik Hamann, Jingrui He, and Hanghang Tong. 2026. Mem-gallery: Benchmarking multimodal long-term conversational memory for mllm agents. *arXiv preprint arXiv:2601.03515*.
- Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. Automanual: Constructing instruction manuals by llm agents via interactive environmental learning. *Advances in Neural Information Processing Systems*, 37:589–631.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. 2026. Cl-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*.
- Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*.
- Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, et al. 2026. A survey of agent memory in the second half: Towards self-evolving and long-horizon agents. *Transactions on Machine Learning Research*.
- Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R Fung. 2026. Xskill: Continual learning from experience and skills in multimodal agents. *arXiv preprint arXiv:2603.12056*.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*.
- Minhua Lin, Hanqing Lu, Zhan Shi, Bing He, Rui Mao, Zhiwei Zhang, Zongyu Wu, Xianfeng Tang, Hui Liu, Zhenwei Dai, et al. 2026a. Position: Agentic evolution is the path to evolving llms. *arXiv preprint arXiv:2602.00359*.
- Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, et al. 2026b. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. *arXiv preprint arXiv:2605.30621*.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

- Jiaqi Liu, Zipeng Ling, Shi Qiu, Yanqing Liu, Siwei Han, Peng Xia, Haoqin Tu, Zeyu Zheng, Chihang Xie, Charles Fleming, et al. 2026a. Omnisimplemem: Autoresearch-guided discovery of life-long multimodal agent memory. *arXiv preprint arXiv:2604.01007*.
- Zewen Liu, Zhan Shi, Yisi Sang, Bing He, Minhua Lin, Tianxin Wei, Dakuo Wang, Benoit Dumoulin, Wei Jin, and Hanqing Lu. 2026b. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams. *arXiv preprint arXiv:2606.01770*.
- Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. 2026. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.
- Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenna Jitsev, Di Lu, Orfeas Menis Mastromichalakis, and 66 others. 2026. [Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces](#). *Preprint*, arXiv:2601.11868.
- Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjuan Jiang. 2026. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*.
- Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. 2026. Code as agent harness. *arXiv preprint arXiv:2605.18747*.
- Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chunliang Li, Yizhu Jiao, Kaiwen Zha, et al. 2026. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*.
- Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. 2025. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.
- Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. 2026. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *arXiv preprint arXiv:2601.22758*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.
- Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*.
- Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2025. Dynamic cheat-sheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*.
- Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.
- Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. 2026. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*.
- Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. 2026. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.
- Tianxin Wei, Naveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. 2025. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*.
- Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiao Zhao, Haifeng Chen, et al. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*.
- Wujiao Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

- Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, et al. 2026. Autoskill: Experience-driven life-long learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ -bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL <https://arxiv.org/abs/2406.12045>, 1(3).
- Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. 2025. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*.
- Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026a. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*.
- Kangning Zhang, Shuai Shao, Qingyao Li, Jianghao Lin, Lingyue Fu, Shijian Wang, Wenxiang Jiao, Yuan Lu, Weiwen Liu, Weinan Zhang, et al. 2026b. Mmskills: Towards multimodal skills for general visual agents. *arXiv preprint arXiv:2605.13527*.
- Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. 2025. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.
- Wanjuan Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731.
- Shuyan Zhou. 2026. [Webarena-infinity: Generating browser environments with verifiable tasks at scale](https://shuyanzhou.com). shuyanzhou.com.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

Appendix

A Benchmark Details

Table 5 summarizes the benchmark settings. We use the same online protocol across benchmarks: tasks are processed as a stream, the solver uses the current harness, and the evolver updates the harness after observing task outcomes.

Table 5: Benchmark statistics and evaluation signals.

Benchmark	#Tasks	Domain	Evaluation signal
WebArena Inf.	80	Web navigation	State verifier
Terminal-Bench	89	CLI / scripting	Docker verifier
SWE-bench Lite	300	Software eng.	Unit tests
CL-Bench	1899	Adaptive reasoning	Rubric judge
TAU-Bench	165	Tool use	State verifier

WebArena Infinity. We use 80 hard tasks from the email-centered “superhuman-general” setting and related web applications. Tasks require stateful browser interaction, such as creating labels, changing settings, using email filters, or coordinating calendar/email information. A programmatic verifier checks the final application state. This benchmark is useful for studying whether skills capture UI procedures.

Terminal-Bench v2. We evaluate 89 tasks across software engineering, data processing, scientific computing, security, system administration, and related command-line domains. The solver works in an isolated shell environment and submits final artifacts or answers inside a task-specific Docker image. Final success is determined by the benchmark’s executable verifier, typically a hidden or mounted test script run inside the container.

SWE-bench Lite. We use 300 real GitHub issues from 12 Python repositories. The solver edits code and the final patch is evaluated by repository tests.

CL-Bench. CL-Bench contains 1899 adaptive reasoning tasks across Domain Knowledge Reasoning, Empirical Discovery & Simulation, Procedural Task Execution, and Rule System Application. Each task is graded with rubric criteria.

TAU-Bench. TAU-Bench contains 165 customer-service tool-use tasks across airline and retail domains. Evaluation checks the final database state.

B Harness Artifacts Examples

This section shows representative learned artifacts without enumerating the learned inventory. For

task-level comparison, we align the same task identifiers between the no-evolve and evolved runs. We only call a case a direct improvement when the baseline run fails and the evolved run succeeds. This is not a single-skill causal ablation: several skills may be injected together, and some benchmarks do not record injected skills in every result file. The examples therefore use a structured failure/success format: task, verifier signal, trajectory summary, learned skill content, and a short take-away.

WebArena: verified setting

- **Task:** task_h10; create “Legal” and disable “Support Ticket”.
- **Baseline:** verifier found “Support Ticket” still enabled, despite the trajectory claiming both subgoals were complete.
- **Evolved:** verifier passed; the trajectory returned to the Auto Labels list and checked the existing label state.
- **Learned skill:** toggle an existing auto label off; do not delete it; verify creation and disabling independently.

Core point. The harness learned a concrete UI distinction: create-new-label and disable-existing-label are separate operations.

WebArena: grounded recipients

- **Task:** task_h65; reply to Kevin Zhao with CTO and VP Engineering on CC.
- **Baseline:** sent to plausible names but wrong domains (acme.com instead of acmecorp.com).
- **Evolved:** verifier passed with the required role-derived recipients in CC.
- **Learned skill:** prefer authoritative role sources and preserve exact email addresses across apps.

Core point. The harness learned identity grounding, not merely email composition.

Terminal-Bench: executable recovery

- **Task:** adaptive-rejection-sampler; pass the Docker verifier for an R sampler.
- **Baseline:** formal tests failed after repeated reruns of the same R test command.
- **Evolved:** passed after diagnosing timeout, missing dependency, formal-test, and sorted-vector failures.
- **Learned skill:** after repeated similar failures, stop, shrink the case, build a minimal reproducer, then add complexity back.

Core point. The harness learned a recovery policy for executable feedback loops.

SWE-bench: test-grounded repair

- **Task:** django__django-11133; fix Django response handling without regressing existing behavior.
- **Baseline:** timed out after source search and failed test attempts.
- **Evolved:** passed FAIL_TO_PASS: 1/1 and PASS_TO_PASS: 64/64.
- **Learned skill:** read complete failing-test assertions, match exact expected behavior, then run targeted tests before submission.

Core point. The harness learned to anchor patches in executable test expectations.

TAU-Bench: complete tool execution

- **Task:** retail_task_5; exchange multiple items from one order.
- **Baseline:** database verifier failed, consistent with partial or wrong-tool execution.
- **Evolved:** verifier passed after retrieving the order and applying all requested item changes.
- **Learned skill:** enumerate every requested operation, choose the tool by order status, and verify the response covers the full list.

Core point. The harness learned complete tool-call execution.

C Which Tasks Are Easier to Improve?

The most informative pattern is not the total number of fail-to-pass cases, but the kind of task that turns around after harness evolution. Table 6 groups matched fail-to-pass cases by their observable failure mode and gives representative task identifiers from the runs in outputs.

D What the Harness Actually Learns

Figure 7 shows real skill excerpts selected with matched fail-to-pass support. The evidence line reports the relevant turnaround family in the runs, while the rule line gives the core learned behavior.

E Prompt Templates

The prompts follow the same context-to-harness compilation logic across experiments: the solver reflects on an execution context to propose candidate lessons, and the evolver curates these candidates against the existing harness. Below, we provide representative prompt templates that capture this common proposal-and-curation process.

These templates are explicit method components and provide an inductive bias toward con-

cise, trigger-conditioned, and reusable skill representations. The resulting skill structure should therefore not be interpreted as emerging without guidance. In a free-form proposal variant without the prescribed schema, we observe more verbose and redundant entries, more frequent harness edits, and weaker downstream performance, suggesting that lightweight structural guidance improves the efficiency and stability of experience compilation.

E.1 Solver-Side Proposal Prompt

Proposal prompt, compressed

Inputs: evaluation result, verifier details or rubric feedback, trajectory signals, compressed trajectory, and related existing skills.

Analyze the execution. For each distinct issue, identify what went wrong and the concrete missing action, command, tool call, navigation step, or domain rule.

Propose a reusable skill. Choose a broad topic, decide NEW, ENHANCE, or NONE, write a trigger-style description, and provide short bullet-point content with techniques and gotchas.

Filter aggressively. Skip generic advice, basic tool usage, exact task replay, and skills that would not help unseen tasks.

E.2 Evolver-Side Curator Prompt

Curator prompt, compressed

Inputs: topic, current skill library, budget, and proposals from the latest batch.

For each proposal, choose ACCEPT, MERGE, or SKIP. Prefer merging over duplication, respect the budget, require a clear trigger description, keep content short and actionable, and apply a generalizability test such as usefulness for multiple unseen tasks.

E.3 General Skill Curator

General skill curator, compressed

Analyze repeated patterns across tasks rather than within a single topic. Create or update a general skill only when a pattern appears across multiple contexts. General skills must avoid context-specific references and encode procedures that can guide planning, verification, recovery, or tool use across tasks.

F Experimental Details

All experiments use the same open-source pipeline with benchmark-specific adapters. WebArena uses browser workers with persistent application state. Terminal-Bench uses task-specific Docker images and executable verifiers. SWE-bench uses isolated repository environments and unit-test evaluation. TAU-Bench uses the official tool-use environment

Table 6: Improvement hotspots from matched baseline-fail/evolved-pass cases.

Benchmark	Easier-to-improve tasks	Representative cases	Observed execution change
WebArena	Stateful UI tasks with exact recipients, labels, settings, and mail state.	h10: disable an existing auto label; h65: preserve corporate email domains; h73: archive all qualifying read-important emails.	Evolved runs more consistently verified the final application state after completing UI operations.
Terminal-Bench	Tasks with executable feedback loops and recoverable failures.	adaptive-rejection-sampler, dna-insert, build-pmars, regex-chess.	Evolved runs more often used executable feedback to isolate failures, revise the solution, and verify artifacts before submission.
SWE-bench Lite	Mature repositories with stable conventions, especially Django and Sympy.	django__django-11133, django__django-17087, sympy__sympy-19254.	Evolved runs more often inspected failing tests and repository conventions before editing and validating patches.
TAU-Bench	Retail exchanges/returns and airline reservation edits with multiple constraints.	retail_task_5, retail_task_91, airline_task_13.	Evolved runs more consistently completed all requested tool operations and verified the resulting state.
CL-Bench	Rule-application, procedural, and domain-knowledge contexts with local rubrics.	Context categories with many turnarounds include Rule System Application, Procedural Task Execution, and Domain Knowledge Reasoning.	Evolved runs more consistently followed task-specific extraction rules, role boundaries, output schemas, and calculation conventions.

and verifier. CL-Bench uses a rubric-based judge pipeline.

The harness is stored as Markdown skill files with lightweight YAML metadata, making the learned artifact inspectable and easy to transfer across runs. Each entry contains a trigger describing when it should be retrieved, a short actionable rule or procedure, optional evidence linking it to prior executions, and a scope indicating whether it is intended as cross-task guidance or task-type guidance. For harness selection, we use Claude Sonnet 4.5 across all experiments to retrieve relevant skills from the current harness before task execution.

Unless otherwise specified, we use a batch size of 16. We set the maximum number of general skills and the maximum number of skills under each task-type topic to 5. This budget encourages the evolver to merge overlapping guidance and avoid accumulating overly detailed or task-specific records.

F.1 Additional Robustness Checks

Task-order robustness. To examine whether online improvement depends on a particular task ordering, we rerun CL-Bench with three task-order seeds using Opus 4.7. Evo-Harness improves over No-Evolve in all three orders, obtaining 34.5, 34.1, and 34.9, respectively.

Static guidance. To separate online evolution from the benefit of structured guidance alone, we construct fixed human-written skills for TerminalBench-2 using only the public benchmark description and tool documentation. Under the same skill format, retrieval mechanism, injection location, and token budget, static guidance improves No-Evolve from 62.92 to 65.17, while Evo-Harness reaches 73.03.

Cross-benchmark transfer. We additionally freeze the general skills evolved on TerminalBench-2 and apply them to SWE-bench Lite without target-stream updates. The solver improves from 63.67 to 64.67. This modest gain suggests that general operational guidance can transfer across benchmarks, while larger gains primarily arise from task-type-specific adaptation.

G Potential Risks

EVO-HARNESS evolves skill libraries that are injected into agent prompts as procedural guidance. The learned skills are designed to capture task-solving patterns such as navigation flows, tool-use procedures, verification steps, and recovery strategies. In our experiments, all benchmarks are run in sandboxed environments with no access to real user data, external accounts, or production systems. Therefore, the evaluated setting poses

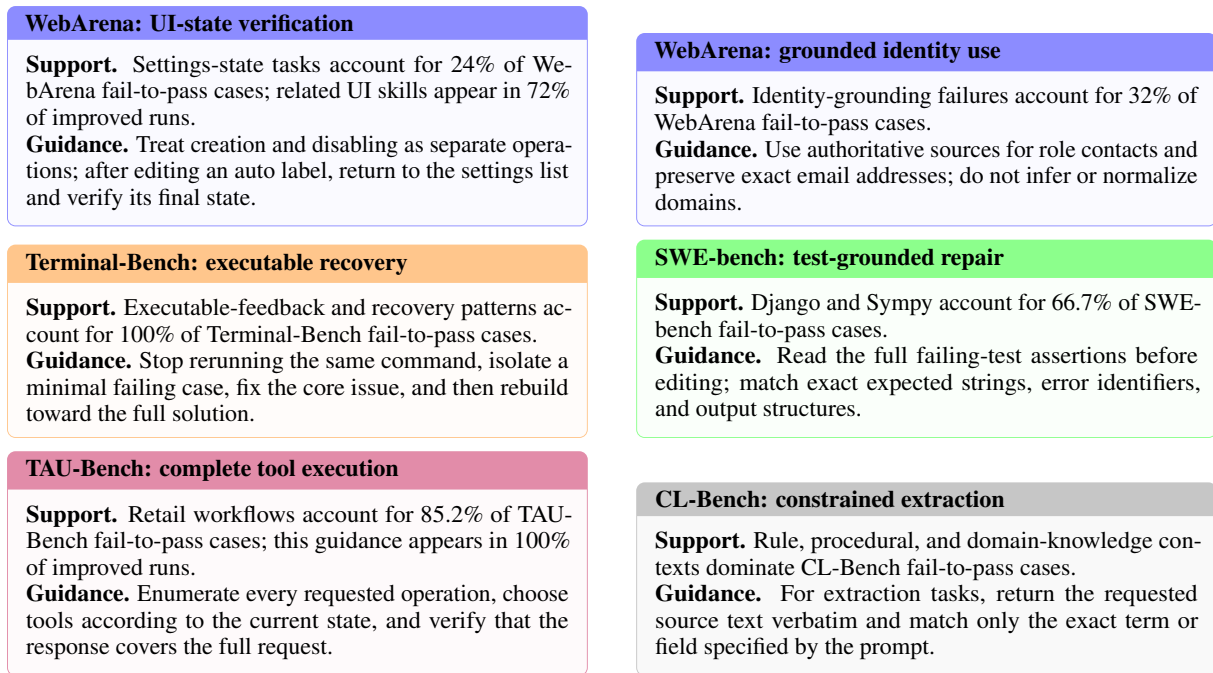


Figure 7: Representative learned harness excerpts across benchmarks. Each card pairs a recurring fail-to-pass pattern with the corresponding evolved guidance. Percentages summarize matched baseline-fail/evolved-pass cases and contextualize the examples rather than establish single-skill causality.

minimal practical risk beyond standard benchmark execution.

H Use Or Create Scientific Artifacts

Our work uses five established public benchmarks: WebArena-Infinity, TerminalBench2, SWE-bench Lite, CL-Bench, and TAU-Bench. We do not modify the benchmarks themselves. We use foundation models (Claude and other models) via their standard AWS Bedrock APIs.

H.1 Cite Creators Of Artifacts

All benchmarks and models are properly cited in the main text.

H.2 Discuss The License For Artifacts

All benchmark resources used in this work are publicly available under their respective research or open-source licenses. In particular, WebArena Infinity and TAU-Bench are released under MIT licenses, WebArena is released under Apache-2.0, and CL-Bench is released under a custom evaluation-only license. Our pipeline code will be released under the MIT License.

H.3 Artifact Use Consistent With Intended Use

All benchmarks are used for their intended purpose of evaluating agent capabilities. The foundation models are used via their standard APIs in accordance with their terms of service.

H.4 Data Contains Personally Identifying Info Or Offensive Content

The benchmarks use synthetic or sanitized scenarios. SWE-bench uses public GitHub issues. To our knowledge, none contain personally identifying information or offensive content.

I Computational Experiments

I.1 Compute Budget

The full experimental suite is computationally expensive because it covers multiple benchmarks, solver models, evolution settings, and ablation configurations. Across all main experiments and diagnostic runs, the total API cost was on the order of \$100K. For a single benchmark configuration, we typically run 16 parallel workers. Depending on benchmark complexity, task timeout, and environment overhead, one full run takes approximately 2–8 hours of wall-clock time. Web and tool-use benchmarks are usually faster, while

software-engineering and command-line tasks tend to take longer because they require repository setup, Docker execution, test running, or verifier calls.

I.2 Experimental Setup And Hyper-params

Key hyperparameters are described in Appendix F. We use fixed random seeds (42) for task shuffling to ensure reproducibility.

I.3 Descriptive Statistics

We report pass rate (%) as the primary metric across all benchmarks. For statistical reliability, we report results over the full task set for each benchmark rather than subsampling. The performance is averaged over three runs.

I.4 Parameters For Packages

The project primarily used AWS Bedrock API for foundation model inference, together with standard AWS infrastructure services such as EC2, S3, EBS/FSx, CloudWatch, IAM, and VPC. The software environment used Python 3.10+, boto3 1.34+, boto-core 1.34+, PyTorch 2.0+, CUDA 12.0+, Docker, and Ubuntu 24.04.

J AI Assistants In Research Or Writing

This paper studies LLMs as the research object and uses frozen LLM backbones for experimental inference. These models are not used to create new benchmark labels or to fabricate experimental results. Reported numerical results come from the experimental code and are checked by the authors.

AI assistant tools were used for polishing manuscript. They were not used to decide scientific claims, select reported results, or replace author verification.