Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Evolver: Turning Agent Exploration into Evolving Capabilities

Haoyang Zhang
17@evomap.ai

EvoMap
Qumeng Sun
EvoMap
Xiao Zhang
EvoMap
Xiang Liu
EvoMap
Zhidong Fan
EvoMap
Junjie Wang
EvoMap https://orcid.org/0000-0001-9869-7085

Method Article

Keywords: LLM Agents, Agent Evolution, Experience-Driven Learning, Procedural Knowledge, Experience Reuse, Agent Memory, Self-Improving Agents

Posted Date: November 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11628893/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Evolver: Turning Agent Exploration into Evolving Capabilities

Haoyang Zhang[†] **Qumeng Sun** **Xiao Zhang** **Xiang Liu**
Zhidong Fan **Junjie Wang**

Infinite Evolution Lab, EvoMap

https://github.com/EvoMap/evolver

Abstract

Agent exploration produces valuable procedural knowledge, yet much of this experience remains confined to transient execution traces and is not systematically reused. We introduce `Evolver`, an experience-driven framework that transforms agent exploration into persistent, reusable capabilities without updating foundation model parameters. Built around structured `Gene` assets, `Evolver` integrates experience acquisition, retrieval, and feedback-guided evolution to support experience-conditioned execution across existing agent harnesses. We evaluate `Evolver` using Claude Code and Codex, two foundation model configurations, and four benchmarks: Terminal-Bench 2.1, SWE-bench Multilingual, SWE-rebench, and GAIA. On a selected diagnostic set of 320 paired executions, `Evolver` improves task success from 83.13% to 87.50% (+4.38 percentage points) while reducing recorded task-execution token consumption by 24.24%. Furthermore, among 265 pairs successful under both conditions, token consumption decreases by 17.57%, demonstrating efficiency benefits even when task success is preserved. These findings suggest that persistent procedural experience can improve both task effectiveness and execution efficiency, providing a practical foundation for transforming agent exploration into evolving capabilities.

1 Introduction

Large language model (LLM) agents have progressed from language generation to multi-step problem solving through tool use and environmental interaction, with growing applications in software engineering and other complex tasks [1, 2]. Executing these tasks often requires extensive environment exploration, tool invocation, failure recovery, and outcome verification, incurring substantial computational and interaction costs. Yet the procedural knowledge acquired through these interactions is often confined to individual executions, leaving subsequent tasks to rediscover useful strategies and repeat similar exploratory efforts. This raises a fundamental question: *How can the computational effort invested in past exploration be transformed into reusable experience that continuously improves future agent capabilities?*

Recent coding agents, such as Codex [3] and Claude Code [4], have demonstrated the growing potential of autonomous software development. Meanwhile, `Skill`-based approaches enable agents to reuse packaged instructions, workflows, and domain-specific resources [5], while other methods leverage historical trajectories, execution feedback, and evolving contexts to improve agent behavior without updating model parameters [6–8]. More recently, research on strategy genes has shown that compact, control-oriented `Gene` representations can provide more effective experience reuse and iterative refinement than documentation-oriented `Skill` representations in controlled scientific coding

[†]Corresponding Author. `17@evomap.ai`

settings [9]. Despite these advances, a central systems challenge remains: how to transform task-specific exploration into persistent, reusable strategy assets that can be integrated with existing agents and progressively refined through execution feedback. Moreover, accumulating experience does not necessarily improve performance, as irrelevant or outdated strategies may introduce redundant computation or interfere with otherwise successful executions.

To address this challenge, we introduce `Evolver`, an experience-driven agent evolution system built around the principle of *Exploration-to-Evolution*: transforming knowledge acquired through task execution into persistent strategy assets that can guide subsequent decisions and support iterative adaptation. Building on the `Gene` representation, `Evolver` encodes reusable procedures and their applicability conditions as structured `Gene` objects, associates them with concrete execution cases through Capsules, and maintains event, provenance, and review records to support experience management. Through integration with existing agent harnesses, `Evolver` makes historical experience available during subsequent task execution without modifying the underlying model parameters. Rather than treating execution histories as passive records, `Evolver` organizes procedural knowledge into reusable assets that can be selected, applied, and maintained in response to new execution feedback. This design establishes a mechanism through which previously invested exploration can contribute to future agent behavior, providing a foundation for experience-driven capability evolution.

We evaluate `Evolver` across two agent harnesses, Claude Code [4] and Codex [3], two foundation model configurations, and four benchmarks: Terminal-Bench 2.1 [10], SWE-bench Multilingual [11], SWE-rebench [12], and GAIA [13]. On a selected diagnostic set of 320 paired executions, `Gene` augmentation improves the overall task success rate from 83.13% to 87.50% (+4.38 percentage points), with gains observed under both Claude Code (+2.50 points) and Codex (+6.25 points). These improvements vary across models and benchmarks, with nine of the 16 evaluated configurations exhibiting higher task success. Beyond task effectiveness, `Evolver` reduces recorded task-execution token consumption by 24.24% overall. Notably, among 265 pairs that succeed both with and without `Evolver`, token consumption decreases by 17.57%, indicating that historical experience can improve execution efficiency even when task success is already achieved. Representative cases in terminal operations and software repair further illustrate these efficiency gains through fewer model invocations and lower token usage without compromising task success. Together, these findings highlight the complementary benefits of reusable procedural experience for task effectiveness and execution efficiency, positioning `Evolver` as a practical systems approach to transforming agent exploration into persistent, reusable capabilities without updating foundation model parameters.

2 Related Work

2.1 Procedural Experience Representation and Reuse

Agent interactions generate procedural knowledge through exploration, tool use, and execution feedback, motivating approaches that reuse historical experience without updating model parameters. Early methods extract reusable insights from successful and failed trajectories [6, 14], while subsequent studies organize such knowledge into workflows, adaptive memory, and procedural guidance [15–18]. Complementary approaches package reusable procedures as executable skills or structured instructions [19–21], whereas Agentic Plan Caching [22] reuses adaptable planning templates to reduce repeated computation. Recent studies further highlight that experience utility depends on representation quality and contextual applicability, as inappropriate skills may increase execution costs or degrade performance [23, 24].

More closely related, the study [9] demonstrates the advantages of compact, control-oriented `Gene` representations over documentation-oriented `Skill` packages for experience reuse and iterative refinement. Building on this representational foundation, `Evolver` operationalizes procedural knowledge as persistent strategy assets associated with execution cases, provenance, and lifecycle states, enabling experience reuse within existing agent systems.

2.2 Experience-Driven Agent Adaptation and Evolution

Beyond experience reuse, recent research investigates how execution feedback can progressively improve agent behavior. GEPA [7] and Agentic Context Engineering [8] evolve prompts and contextual guidance through reflection and feedback, while EvolveR [25] combines experience-

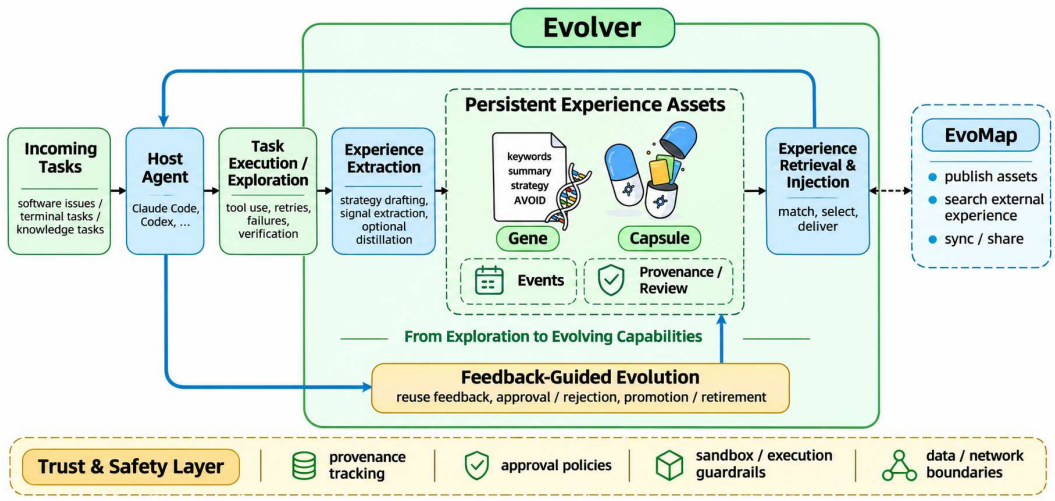


Figure 1: Overview of `Evolver` and its Exploration-to-Evolution framework, illustrating experience acquisition, persistent strategy assets, experience-conditioned execution, feedback-supported evolution, EvoMap connectivity, and operational trust boundaries.

derived principles with reinforcement learning. Other approaches support continual adaptation through procedural memory refinement, skill evolution, and experience governance [26–29]. At the implementation level, Godel Agent [30] and Darwin Godel Machine [31] pursue self-improvement through modifications to agent code.

These studies demonstrate complementary pathways for agent evolution, spanning prompts, memory, skills, model policies, and executable programs. In contrast, `Evolver` focuses on evolving external procedural experience while preserving the underlying foundation model and existing agent harnesses. Its *Exploration-to-Evolution* design connects experience acquisition, structured strategy reuse, and feedback-supported lifecycle management, allowing historical exploration to inform subsequent execution. We examine the conditional benefits and computational trade-offs of experience reuse, recognizing that accumulated experience does not necessarily yield consistent improvements.

3 `Evolver`: From Exploration to Evolution

3.1 Design Principles and System Overview

Problem Formulation. Let π_θ denote a fixed foundation-model policy, h an existing agent harness, and x_t the task encountered at step t. Without updating model parameters, `Evolver` maintains an external experience state $\mathcal{E}_t$ to inform subsequent execution. The interaction is formulated as

$$\begin{aligned} g_t &= \mathcal{R}_h(x_t, \mathcal{E}_t), \\ \tau_t &= \mathrm{Exec}_{\pi_\theta, h}(x_t; g_t), \\ \mathcal{E}_{t+1} &= \mathcal{U}(\mathcal{E}_t, \tau_t, f_t), \end{aligned} \tag{1}$$

where $\mathcal{R}_h$ retrieves applicable experience, τ_t denotes the resulting execution trajectory, and f_t represents available execution feedback. The update operator $\mathcal{U}$ abstracts experience acquisition and lifecycle operations, which depend on the enabled system components.

Design Principles. `Evolver` follows the *Exploration-to-Evolution* principle, which transforms knowledge acquired through agent exploration into persistent strategy assets that can support future adaptation. This principle is realized through three complementary designs. First, *Experience Externalization* separates reusable procedural knowledge from individual execution trajectories, enabling its persistent representation and management. Second, *Experience-Conditioned Execution* makes relevant strategies available to existing agents without modifying their underlying model parameters or replacing their execution harnesses. Third, *Feedback-Guided Evolution* supports the assessment and maintenance of accumulated experience through execution feedback and lifecycle operations. Together, these designs establish a pathway for preserving the knowledge value of historical exploration and making it available to subsequent agent executions.

System Architecture. As illustrated in Figure 1, `Evolver` integrates experience acquisition, persistent asset management, retrieval, and feedback-supported evolution into a unified system. Historical agent interactions provide execution traces from which reusable strategies are extracted through signal analysis, automatic drafting, or optional model-based distillation. The resulting experience is organized into structured `Gene` objects, associated `Capsule` records, and supporting event, provenance, and review information. During subsequent tasks, relevant experience is retrieved and delivered through hooks, MCP interfaces, or host-specific integrations to condition agent behavior. `Evolver` further connects to EvoMap [1] for network-based experience publishing, discovery, and retrieval. Provenance tracking, review mechanisms, and execution safeguards provide operational controls whose enforcement depends on the integration pathway and deployment environment. The following sections detail these components and their roles in transforming historical exploration into reusable and evolving agent capabilities.

3.2 Experience Representation and Asset Management

Structured Strategy Representation. To preserve the knowledge acquired through historical exploration, `Evolver` represents reusable procedural knowledge as structured strategy assets rather than retaining complete execution trajectories as the primary unit of reuse. Building on the strategy-gene representation [9], each `Gene` g_i is characterized by

$$g_i = (s_i, \mathcal{S}_i, \mathcal{P}_i, \mathcal{M}_i, \mathcal{V}_i), \tag{2}$$

where s_i summarizes the experience, $\mathcal{S}_i$ specifies procedural steps, $\mathcal{P}_i$ describes applicability conditions, $\mathcal{M}_i$ provides task-matching signals, and $\mathcal{V}_i$ contains optional validation plans. This representation separates reusable operational guidance from task-specific execution details, allowing historical strategies to be retrieved and adapted to subsequent tasks. Importantly, applicability conditions and validation plans provide guidance rather than enforceable execution constraints or guarantees of prior verification.

Complementary Experience Assets. Beyond `Gene` objects, `Evolver` maintains complementary assets that preserve execution context and support experience management. A `Capsule` associates a strategy with a concrete execution case, providing contextual examples that can accompany retrieved strategies. An `AntiGene` captures negative procedural guidance, identifying actions or practices to avoid in subsequent executions. Meanwhile, `Event` records capture asset-related operations, while separate provenance and review records track experience origins, trust status, and approval decisions. Together, these components constitute the persistent experience state $\mathcal{E}_t$, separating reusable strategies, supporting execution cases, and operational metadata.

Persistent Asset Management. `Evolver` maintains experience assets through content-addressed, append-only storage, preserving asset identity independently of subsequent lifecycle decisions. Review and provenance states are maintained in separate records, supporting experience quarantine, approval, rejection, and revocation without overwriting the underlying assets. This separation enables persistent procedural knowledge to coexist with evolving eligibility and trust states, providing a foundation for feedback-driven experience management. The enforcement of these states depends on the corresponding acquisition and retrieval pathways, as detailed in Sections 3.3 to 3.5.

3.3 Experience Acquisition and Consolidation

Experience Sources. `Evolver` treats historical agent interactions as a source of reusable procedural knowledge, capturing not only task outcomes but also the exploration and feedback that precede them. When session ingestion is enabled, runtime adapters collect records from supported agent harnesses, including task context, tool interactions, execution errors, and available outcomes. These records are normalized into bounded experience materials with heuristic sensitive-information redaction. Additional experience can be acquired through explicit asset submissions and external experience repositories, providing complementary sources beyond automatically collected sessions.

Experience Extraction. Given collected experience materials, `Evolver` identifies lightweight signals associated with successful execution, failures, and task difficulty to construct candidate `Gene` objects. The default automated drafting pathway extracts concise procedural hints from assistant utterances and marks them as unproven, rather than assuming that the resulting strategies have been validated. When

[1] https://evomap.ai/

enabled, LLM-based distillation further supports structured experience extraction from execution records, including reusable strategies, associated cases, and negative procedural guidance. These complementary pathways separate low-cost experience collection from more structured synthesis, while leaving the effectiveness of generated strategies subject to subsequent evaluation.

Experience Consolidation. Extracted experience is consolidated into persistent, content-addressed assets, with supporting execution cases retained when available. Automatically drafted and model-distilled candidates are placed under review, whereas certain explicit submission pathways can persist assets directly. Review and provenance records maintain experience eligibility and source information separately from the underlying strategy content, although admission and trust enforcement depend on the acquisition and retrieval pathways. By preserving candidate knowledge independently of its current lifecycle state, `Evolver` establishes a reusable experience base that supports subsequent retrieval and feedback-guided evolution, as detailed in Sections 3.4 and 3.5.

3.4 Experience Retrieval and Agent Integration

Task-Conditioned Experience Retrieval. To reuse historical knowledge, `Evolver` provides multiple retrieval pathways that select experience according to task context and integration requirements. Session-start hooks supply a bounded set of locally eligible `Gene` objects, while optional prompt-time recall matches task descriptions against stored experience signals. MCP interfaces additionally support on-demand experience retrieval, whereas autonomous execution employs a separate heuristic selection mechanism incorporating task relevance, experience quality, and historical reuse feedback. These pathways differ in their ranking strategies, context budgets, and eligibility checks, allowing experience delivery to accommodate different execution settings.

Experience-Conditioned Execution. Retrieved `Gene` objects, together with associated `Capsule` examples when available, are delivered through session hooks, prompt-time injection, MCP tools, or host-specific integrations. Rather than modifying foundation model parameters or replacing agent execution policies, `Evolver` introduces procedural guidance into the existing interaction context, enabling the host agent to interpret and apply historical strategies. This separation between experience management and task execution allows `Evolver` to augment existing agent harnesses while preserving their native reasoning, tool-use, and verification capabilities. Consequently, historical exploration can inform subsequent decisions without requiring additional model training or modifications to the underlying agent architecture.

Networked Experience through EvoMap. Beyond local experience reuse, `Evolver` integrates with EvoMap to support the publication, discovery, and retrieval of experience assets across connected agents. This interface extends the available knowledge beyond individual execution histories, allowing agents to access strategies contributed by other nodes. External assets can be associated with provenance and trust metadata, although direct retrieval and locally imported assets follow different admission and review pathways. Together, local and network-based experience access provides a foundation for broader experience reuse.

3.5 Feedback-Guided Experience Evolution

Execution Feedback. Beyond preserving and reusing historical knowledge, `Evolver` supports experience evolution through feedback collected from subsequent executions. Depending on the execution pathway, feedback includes task outcomes, validation results, reuse events, and interaction signals from agent hosts. Autonomous execution evaluates task outcomes using execution status and available verification procedures, while host integrations can report observed checks and reuse-related signals. These observations provide a basis for experience assessment.

Experience Lifecycle. Rather than treating accumulated experience as permanently applicable, `Evolver` maintains lifecycle states that govern how strategies are retained and reused. Automatically drafted and model-distilled `Gene` candidates initially enter quarantine and can subsequently be approved or rejected through review. An optional probation mechanism further supports promotion based on recorded successes and failures, while autonomous execution incorporates historical reuse signals and failure-based restrictions into candidate selection. Rejection and provenance revocation are recorded separately from the underlying content-addressed assets, preserving their identity while modifying their subsequent eligibility. These mechanisms instantiate the lifecycle operations

underlying $\mathcal{U}$ in Equation (1), allowing experience states to evolve without modifying foundation model parameters.

Trust-Aware Evolution. Because experience may originate from local execution or external contributors, `Evolver` associates assets with provenance and review information to support source-aware management. Imported EvoMap assets can be marked as untrusted until explicitly promoted, while supported execution pathways apply additional validation and operational safeguards. However, admission checks and trust enforcement are pathway-dependent, and some direct retrieval interfaces do not apply the same restrictions. Together, these mechanisms provide a foundation for feedback-guided experience adaptation, distinguishing the evolution of reusable strategy states from an assumption of uniformly improving agent performance.

4 Experimental Evaluation and Analysis

4.1 Experimental Setup

Benchmarks and Agent Configurations. We evaluate `Evolver` on four benchmarks covering diverse agent capabilities: Terminal-Bench 2.1 [10], SWE-bench Multilingual [11], SWE-rebench [12], and GAIA [13]. Terminal-Bench 2.1 evaluates terminal-based engineering tasks, SWE-bench Multilingual and SWE-rebench assess software issue resolution, and GAIA examines multi-step information seeking and reasoning. We consider two agent harnesses, Claude Code and Codex, each paired with two foundation model configurations, `deepseek-v4.1-flash` [32] and `gpt-5.6-sol` [33]. For each benchmark, we prioritize the first 20 tasks in their original order. Tasks yielding empty responses due to malformed outputs, timeouts, or execution failures are replaced sequentially with subsequent tasks. We ensure that all harness–model configurations are evaluated on the same 20 tasks per benchmark, yielding 16 configurations in total. All experiments use `Evolver` v2.0.33.

Experience Construction and Paired Evaluation. We adopt a three-stage *Baseline–Extraction–Gene* (B–E–G) protocol to examine the effects of historical experience reuse. In the *Baseline Execution* (B) stage, each agent executes the original task using its native tools and built-in skills without additional `Gene` injection. In the *Experience Extraction* (E) stage, reusable strategies are distilled from valid baseline trajectories, including both successful and unsuccessful executions, using a model configuration matched to the corresponding evaluation setting. Experience pools are isolated by evaluation batch, benchmark, harness, and model, and candidate assets are frozen before augmented execution. In the *Gene-Augmented Execution* (G) stage, the agent receives the original task together with selected `Gene` guidance in a fresh session with the task environment restored. Gene selection prioritizes experience derived from the same task, followed by other matching candidates within the isolated pool, with an injection budget of 12,000 bytes. Both B and G retain the same foundation model and harness configuration and are evaluated using the corresponding native benchmark grader. This protocol assesses historical experience reuse rather than generalization to unseen tasks or long-term online evolution.

Evaluation Criteria and Metrics. We analyze 320 paired observations, retaining 20 tasks per configuration with valid outcomes and complete token records for both B and G, excluding executions interrupted by timeouts or token limits. The final task selection additionally considers observed outcome improvements and token usage, making the resulting set diagnostic rather than a random sample of the full benchmarks. We report task success rates and percentage-point differences, together with recorded task-execution tokens and model invocation counts. Aggregate token reduction is computed from total usage across paired executions rather than averaging individual reduction ratios. We further analyze pairs that succeed under both conditions to assess resource changes while preserving task success. Token accounting includes recorded model input and output usage but excludes experience extraction and other overhead outside the final task executions.

4.2 Overall Task Performance

Overall Effectiveness. As shown in Figure 2, incorporating historical `Gene` experience increases the overall success rate from 83.13% to 87.50%, an improvement of 4.38 percentage points across 320 paired evaluations. These results demonstrate the potential of reusable procedural experience to improve task completion within the selected evaluation setting.

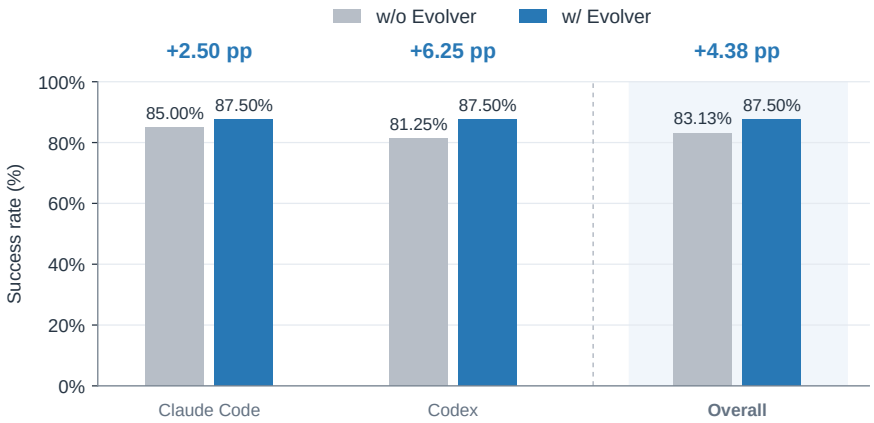


Figure 2: Overall task success with and without `Evolver` across two agent harnesses and the combined evaluation set.

Performance across Agent Harnesses. Both evaluated harnesses exhibit improvements when augmented with `Evolver`. Claude Code improves from 85.00% to 87.50% (+2.50 percentage points), while Codex increases from 81.25% to 87.50% (+6.25 percentage points). These consistent gains across both harnesses highlight the applicability of experience-conditioned execution to different agent runtimes without modifying their underlying model parameters.

4.3 Performance across Models and Benchmarks

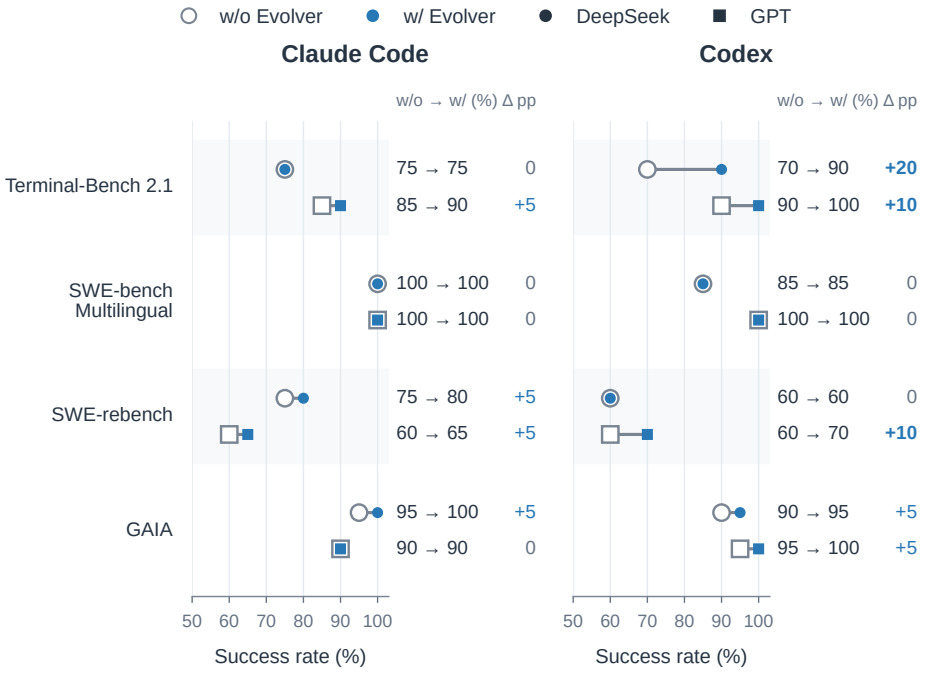


Figure 3: Task success across agent harnesses, foundation models, and benchmarks, with 20 paired evaluations per configuration.

Model-Dependent Performance. As shown in Figure 3, the benefits of experience reuse vary across foundation models and agent harnesses. Among the 16 evaluated configurations, nine exhibit improved task success, while seven remain unchanged. Under Codex, DeepSeek achieves the largest improvement on Terminal-Bench 2.1, increasing from 70% to 90% (+20 percentage points), whereas GPT improves by 10 points on both Terminal-Bench 2.1 and SWE-rebench. For Claude Code, both models improve by 5 points on SWE-rebench, while additional gains occur on different benchmarks: GAIA for DeepSeek and Terminal-Bench 2.1 for GPT. These results suggest that the effectiveness of

historical procedural guidance depends on the underlying model and its interaction with the agent harness.

Benchmark-Dependent Performance. Across the evaluated configurations, Terminal-Bench 2.1 exhibits the largest aggregate improvement, increasing from 64/80 to 71/80 (+8.75 percentage points). SWE-rebench and GAIA also improve by 5.00 and 3.75 points, respectively, demonstrating benefits across software engineering and information-seeking tasks. In contrast, SWE-bench Multilingual maintains the same success rate of 96.25% under both conditions. These findings highlight the task-dependent nature of experience reuse, while motivating further analysis of whether historical knowledge can reduce execution costs even when task success remains unchanged.

4.4 Token Efficiency of Experience Reuse

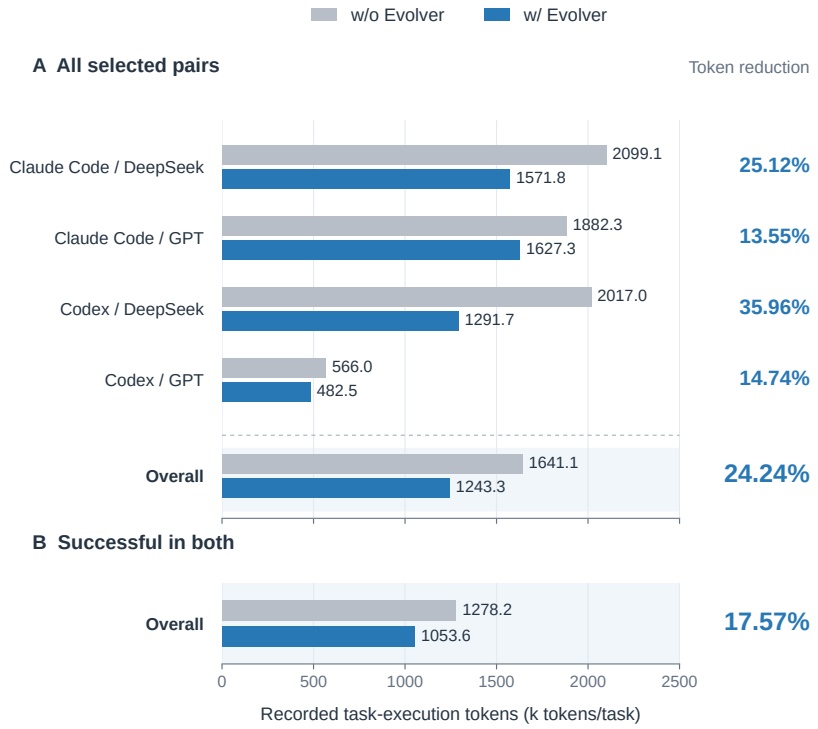


Figure 4: Recorded task-execution token usage with and without `Evolver` across all selected pairs (A) and pairs successful under both conditions (B).

Overall Token Reduction. As shown in Figure 4(A), `Evolver` reduces average task-execution token consumption from 1,641.1K to 1,243.3K, corresponding to an aggregate reduction of 24.24% across 320 paired evaluations. Token usage decreases across all four harness–model configurations, with reductions ranging from 13.55% to 35.96%. The largest reduction occurs with Codex and DeepSeek (35.96%), followed by Claude Code and DeepSeek (25.12%). These results indicate that historical procedural experience can reduce the computational resources required for subsequent agent execution across different configurations.

Efficiency under Preserved Success. We further consider *Successful in Both*, a subset of 265 paired tasks that pass the native benchmark evaluation both with and without `Evolver`. As shown in Figure 4(B), average token usage decreases from 1,278.2K to 1,053.6K, yielding a 17.57% reduction while maintaining successful task completion. This finding highlights the value of reusable experience beyond improving task success, demonstrating its potential to reduce execution overhead for tasks that agents can already solve.

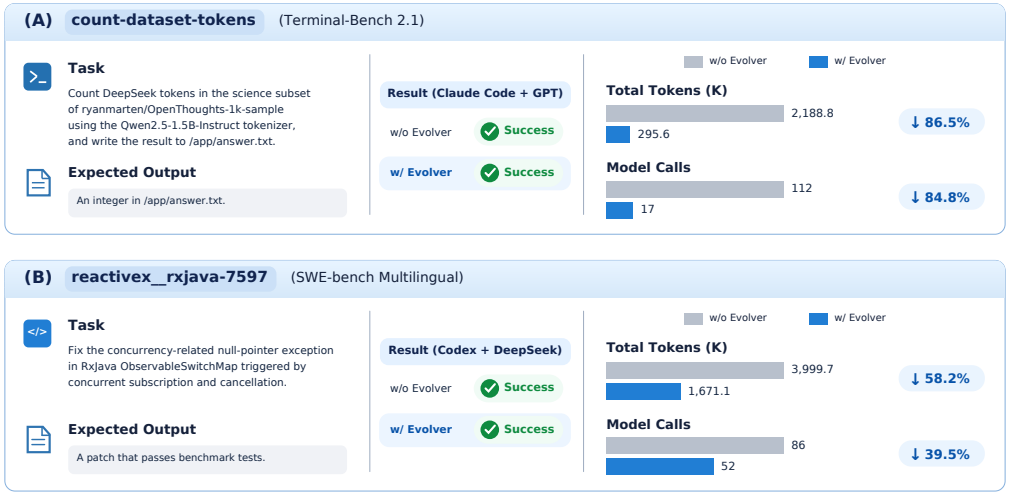


Figure 5: Representative cases showing preserved task success with reduced token consumption and model invocations.

4.5 Case Studies

To complement the aggregate results, Figure 5 presents two representative tasks where historical experience reduces execution resources without changing task success.

Dataset Token Counting. As illustrated in Figure 5(A), this Terminal-Bench 2.1 task requires counting tokens in a specified dataset subset using a designated tokenizer and producing an integer result. With Claude Code and GPT, both executions succeed, while `Evolver` reduces token consumption by 86.5% and model invocations from 112 to 17.

Software Repair. As illustrated in Figure 5(B), this SWE-bench Multilingual task involves resolving a concurrency-related null-pointer exception in RxJava's `ObservableSwitchMap`. With Codex and DeepSeek, both executions pass the benchmark evaluation, while `Evolver` reduces token consumption by 58.2% and model invocations from 86 to 52.

5 Conclusions

In this work, we introduced `Evolver`, a framework for transforming transient agent exploration into persistent, reusable procedural experience. By integrating structured `Gene` representation, experience acquisition, retrieval, and feedback-guided evolution, `Evolver` enables existing agent harnesses to reuse accumulated knowledge without modifying foundation model parameters. Across two agent harnesses, two foundation model configurations, and four benchmarks, our selected diagnostic evaluation shows that experience reuse improves task success by 4.38 percentage points while reducing recorded task-execution token consumption by 24.24%. Notably, token consumption decreases by 17.57% among tasks successfully completed under both conditions, suggesting that reusable experience can improve execution efficiency even when task success remains unchanged. These findings highlight the complementary roles of accumulated procedural knowledge in improving task effectiveness and reducing redundant execution effort. Future work will investigate more reliable experience selection and validation, transfer to unseen tasks, and longitudinal evolution of reusable capabilities. More broadly, `Evolver` provides a systems foundation for turning isolated agent interactions into persistent knowledge that can support continued capability improvement.

References

[1] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024.

[2] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *CoRR*, abs/2409.02977, 2024.

[3] OpenAI. Introducing codex. OpenAI Blog, May 2025. URL https://openai.com/index/introducing-codex/.

[4] Boris Cherny. Claude code: Best practices for agentic coding. Anthropic Engineering Blog, April 2025. URL https://www.anthropic.com/engineering/claude-code-best-practices.

[5] Anthropic. Equipping agents for the real world with agent skills. Anthropic Research Blog, October 2025. URL https://www.anthropic.com/research/equipping-agents-for-the-real-world-with-agent-skills.

[6] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM agents are experiential learners. In *AAAI*, pages 19632–19642. AAAI Press, 2024.

[7] Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: reflective prompt evolution can outperform reinforcement learning. In *ICLR*. proceedings.iclr.cc, 2026.

[8] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y. Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *ICLR*. proceedings.iclr.cc, 2026.

[9] Junjie Wang, Yiming Ren, and Haoyang Zhang. From procedural skills to strategy genes: Towards experience-driven test-time evolution. *CoRR*, abs/2604.15097, 2026.

[10] Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis-Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *ICLR*. proceedings.iclr.cc, 2026.

[11] John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In *NeurIPS*, 2025.

[12] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *NeurIPS*, 2025.

[13] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *ICLR*. OpenReview.net, 2024.

[14] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[15] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *ICML*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

[16] Mirac Suzgun, Mert Yüksekgönül, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *EACL (Volume 1: Long Papers)*, pages 7080–7106. Association for Computational Linguistics, 2026.

[17] Yuli Qiu, Yutong Li, Wei Su, Zeming Liu, Wanxiang Che, Heyan Huang, Haifeng Wang, and Yuang Guo. Lifemem: Enabling lifelong experience reuse for LLM agents. *CoRR*, abs/2609.12655, 2026.

[18] Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *ACL (Findings)*, pages 16803–16822. Association for Computational Linguistics, 2026.

[19] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024, 2024.

[20] Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. Anthropic Research Blog, October 2025. URL https://www.anthropic.com/research/equipping-agents-for-the-real-world-with-agent-skills.

[21] Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. In *ICML*, volume 306, pages 88574–88595, 2026.

[22] Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. In *NeurIPS*, 2025.

[23] Yujian Liu, Jiabao Ji, Li An, Tommi S. Jaakkola, Yang Zhang, and Shiyu Chang. How well do agentic skills work in the wild: Benchmarking LLM skill usage in realistic settings. *CoRR*, abs/2604.04323, 2026.

[24] Gen Dong, Yanjie Gao, Liqun Li, Tianyin Xu, Yu Hua, and Fan Yang. Agent skills can be harmful: An empirical study of skill-induced failures in LLM agents. *CoRR*, abs/2608.11888, 2026.

[25] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. From interactions to principles: Experience-driven self-distillation for evolving LLM agents. In *ICML*, volume 306 of *Proceedings of Machine Learning Research*, pages 137024–137044, 2026.

[26] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *CoRR*, abs/2602.08234, 2026.

[27] Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. Muse-autoskill: Self-evolving agents via skill creation, memory, management, and evaluation. *CoRR*, abs/2605.27366, 2026.

[28] Bo Tang, Yang Zhang, Guomian Zhuang, Wenqiang Wei, Gaoyang Zheng, Lindong Xie, Yanchao Tan, Feiyu Xiong, Qingyu Yang, Edward Chung, and Zhiyu li. From memory to skills: Evidence-grounded co-evolution governance for long-horizon LLM agents. *CoRR*, abs/2607.16621, 2026.

[29] Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *CoRR*, abs/2512.18746, 2025.

[30] 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 *ACL (1)*, pages 27890–27913. Association for Computational Linguistics, 2025.

[31] Jenny Zhang, Shengran Hu, Cong Lu, Robert T. Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *ICLR*. proceedings.iclr.cc, 2026.

[32] DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *CoRR*, abs/2609.19969, 2026.

[33] OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. OpenAI Blog, July 2026. URL https://openai.com/index/gpt-5-6/.