OPENSPACE: SKILL ECOSYSTEMS FOR SELF-EVOLVING AGENT HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills encapsulate reusable procedures imported from communities or abstracted from past executions. However, as libraries expand, overlapping scopes and uneven quality complicate retrieval, while static instructions fail to generalize across varying task contexts. We introduce OpenSpace, a self-evolving agent harness that transforms static skill collections into a dynamic ecosystem continually refined by execution evidence. OpenSpace operates through three coupled mechanisms: evidence-grounded acquisition (abstracting verified methods from raw trajectories), quality-driven maintenance (structurally integrating repairs and derivations while tracking empirical utility), and context-aware activation (dynamically binding optimal skills to the runtime environment). On EvoAgentBench, SkillFlow, and LiveMCPBench, OpenSpace delivers substantial gains in procedural transfer and continual task completion. Strictly without updating the foundation model's weights, it outperforms the strongest non-oracle baselines across all twelve backbone-domain settings on EvoAgentBench (by 5.4–34.5 percentage points) and boosts task completion on SkillFlow from 36.14% to 42.77%. Because OpenSpace evolves the external ecosystem rather than the internal model, it provides a highly effective, training-free paradigm to scale agent capabilities. We have made OpenSpace anonymously available at: https://anonymous.4open.science/r/OpenSpace-ICLR2027-2F8C/.

1 INTRODUCTION

As AI agents increasingly tackle complex, open-world tasks, they rely heavily on persistent skill libraries to encapsulate reusable procedures, operational metadata, and execution resources (Wang et al., 2025; Zhang et al., 2026b; Agent Skills, n.d.). Whether imported from external communities or autonomously abstracted from the agent's own execution trajectories (Liu et al., 2026; Alzubi et al., 2026), these skills establish a foundational procedural knowledge base that persists across tasks and evolves with experience.

However, agents operating within dynamically expanding skill libraries face a critical bottleneck: indiscriminately accumulating procedures does not equate to acquiring actionable capabilities. Specifically, we identify three core challenges in managing this evolving skill ecosystem. First, **contextual mismatch**: a growing library inevitably accumulates skills with overlapping scopes, uneven quality, and conflicting assumptions (Song & Wei, 2026; Liu et al., 2026). Superficial semantic matching fails to determine whether a broad workflow or a highly specialized subprocedure is optimal for the current runtime environment. Second, **ambiguity of attribution**: although execution feedback is essential for continuous improvement (Yang et al., 2026; Tang et al., 2026; Ouyang et al., 2026a), isolating precisely *what* to learn remains complex. For instance, if an agent successfully employs a fallback OCR tool on a scanned invoice after a standard text-extraction skill fails, does this warrant an entirely new skill, a specialized derivation, or a bug fix to the original parent? Furthermore, task success achieved after abandoning a flawed method must not retroactively credit the failed guidance. Third, **ecosystem bloat**: capturing every successful execution trace as a new skill generates redundant scripts, which severely degrades downstream retrieval efficiency and agent reliability. Effective reuse therefore demands a framework that explicitly connects what a procedure prescribes, where it applies, and what its empirical usage has actually established.

To address these coupled challenges, we introduce **OpenSpace**, a self-evolving agent harness designed to transform static skill collections into a persistently maintained, evidence-driven ecosystem. The

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

core philosophy of OpenSpace is to decouple the agent's external procedural knowledge from the internal weights of the foundation model, thereby enabling training-free adaptation. OpenSpace achieves this by explicitly linking reusable procedures to version-specific empirical evidence, using this linkage to strictly govern both structural revisions and subsequent invocations.

Specifically, OpenSpace orchestrates a continuous evolutionary cycle through three integrated mechanisms. First, **evidence-grounded acquisition** evaluates raw execution trajectories to distinguish genuine procedural increments from exploratory detours. Rather than indiscriminate additions, it proposes targeted structural updates: CAPTURE for novel methods, FIX for evidenced defects, and DERIVE for reusable extensions. Second, **quality-driven maintenance** integrates these updates following a rigorous evidence review. It strictly separates the commitment of structural changes from the continuous tracking of empirical utility, ensuring that a skill accrues trust only when its active application directly drives task success. Finally, **context-aware retrieval and activation** dynamically selects optimal skills based on historical quality metrics and current task relevance. Bounded inspection assesses candidate skills before invocation; upon invocation, operational bindings (e.g., event hooks and configuration scopes) (Anthropic, n.d.) configure the selected procedure for the runtime environment.

Our main contributions can be summarized as follows:

- **Evidence-Grounded Skill Acquisition.** We develop an attribution mechanism that utilizes execution traces and source coverage to rigorously distinguish among novel capabilities, supported corrections, and experiences warranting no structural update.
- **Quality-Driven Skill Ecosystem Maintenance.** We introduce a versioned integration pipeline that combines reviewed structural integrations with continuous quality assessment, thereby preserving the exact empirical evidence behind every ecosystem revision.
- **Context-Aware Skill Retrieval and Activation.** We design an adaptive retrieval and activation strategy that connects task relevance and historical usage trust to dynamic runtime binding, enabling agents to leverage maintained skills for both targeted and partial reuse.

2 PRELIMINARY

We consider a self-evolving agent harness that equips a foundation model $\mathcal{M}$ with a persistent skill ecosystem to sequentially solve tasks. Before executing task x_t, we formalize this ecosystem as $\mathcal{E}_t = (\mathcal{S}_t, \mathcal{H}_t)$. The skill library $\mathcal{S}_t$ encapsulates reusable procedural instructions, descriptive metadata, and supporting resources, either imported from external repositories or autonomously abstracted from past executions. The historical context $\mathcal{H}_t$ tracks accumulated execution evidence, quality records, and revision histories.

To solve x_t, the harness dynamically retrieves relevant skills from $\mathcal{S}_t$ and conditions $\mathcal{M}$ on them to fulfill runtime requirements. This execution yields an empirical trajectory ξ_t and environmental feedback y_t, which drive the ecosystem update:

$$\mathcal{E}_{t+1} = \mathcal{U}(\mathcal{E}_t, x_t, \xi_t, y_t). \tag{1}$$

The update operator $\mathcal{U}$ governs the evolutionary dynamics of the harness: it synthesizes new skills, refines existing ones, or exclusively updates evidence records in $\mathcal{H}_t$. Crucially, this mechanism enables iterative evidence accumulation, which limits unnecessary skill additions while grounding structural updates in empirical outcomes.

Our primary objective is to formulate an evolution paradigm without updating the internal weights of $\mathcal{M}$: investigating how the harness can autonomously adapt its external ecosystem to improve downstream performance. Achieving this training-free evolution requires determining **what to acquire**: which empirical execution traces warrant abstraction into reusable skills; **how to maintain**: how accumulated evidence should guide skill repair, derivation, and continuous quality assessment; and **when and how to invoke**: how to adaptively select and activate skills by jointly evaluating task context, runtime conditions, and historical quality records.

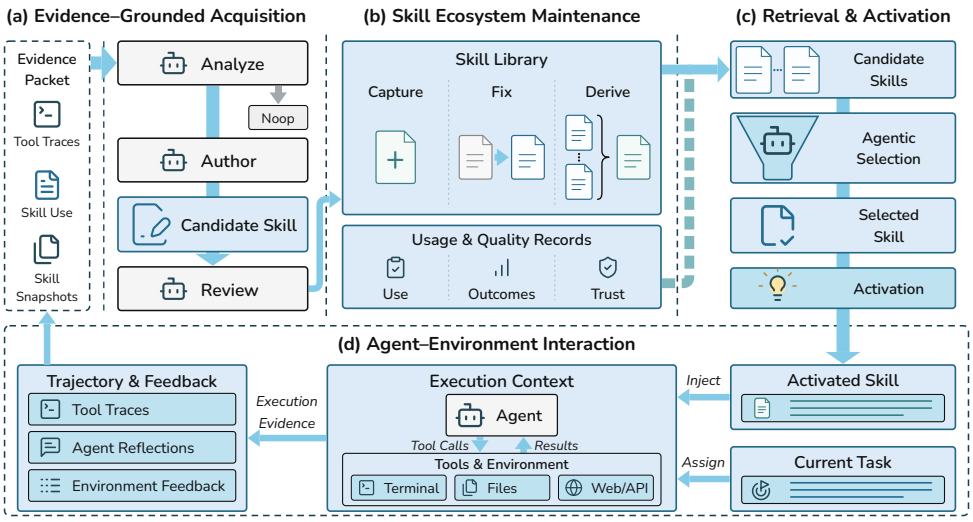

Figure 1: **Overview of OpenSpace.** (a) Execution evidence supports candidate skill analysis, authoring, and review. (b) The ecosystem maintains captured, repaired, and derived skills alongside usage and quality records. (c) Retrieval and agentic selection identify skills for activation. (d) Activated skills guide the agent's interaction with tools and the environment, producing trajectories and feedback for subsequent updates.

3 The OpenSpace Framework

Operating as a self-evolving harness, OpenSpace transforms static skill collections into a persistently maintained, evidence-driven ecosystem. The framework enforces a strict separation among raw observations, their procedural interpretations, and the resulting actionable skills. To bridge these semantic gaps, OpenSpace orchestrates three coupled mechanisms: **evidence-grounded acquisition** distills reusable procedures from noisy execution traces; **quality-driven maintenance** integrates proposed updates topologically while calibrating their empirical utility; and **context-aware retrieval and activation** dynamically selects and binds optimal guidance to the current execution. Conceptually, acquisition and maintenance instantiate the ecosystem update in Equation 1, while runtime activation generates the ensuing observations to drive the next evolutionary cycle. This closed-loop adaptation develops the external skill ecosystem while keeping the foundation model's parameters strictly frozen. Figure 1 summarizes this lifecycle.

A Unified Skill Representation. Imported community skills and autonomously generated procedures share a unified discovery and invocation interface. Each skill adheres to the Agent Skills directory format (Agent Skills, n.d.), encapsulating procedural instructions, descriptive and operational metadata, and supporting resources. Crucially, associated records strictly link each version to its parent revisions, source evidence, and historical usage. This decoupling ensures that the directory supplies generalized, reusable guidance, while the underlying records support data-driven revisions without hardcoding past trajectories into future instructions.

3.1 Evidence-Grounded Skill Acquisition

Distilling High-Signal Evidence. Execution records continuously accumulate in $\mathcal{H}$. To extract high-signal knowledge from this stream, each analysis operates on a bounded evidence packet containing observations from the trajectory ξ_t, available feedback y_t, and frozen copies of the exact skill contents supplied during execution. Missing results are treated as explicit knowledge gaps rather than inferred outcomes. A tool call and its direct result constitute **one indivisible observation**—preventing duplicate validation credit—while the frozen sources preserve the precise guidance available to the agent at execution time.

The analyzer first distinguishes passive skill availability from active application, and subsequently evaluates both instruction defects and procedural coverage. **The structural relationship to the used guidance dictates the specific update trajectory**: an attributable instruction defect triggers a FIX; a reusable operation, branch, or check that exceeds the source's coverage motivates a DERIVE; and a

newly executed method lacking an appropriate inheritance source necessitates a CAPTURE. Coverage is rigorously assessed by determining whether the supplied procedures and conditions suffice under ordinary input substitution. If the evidence supports neither a verifiable defect nor a procedural increment, the analysis gracefully yields a NOOP.

Synthesizing Reusable Procedures. Before generation, admission protocols verify that all cited observations and required source versions remain accessible. The current configuration additionally requires a validated, successful source attempt for CAPTURE; eligible failed attempts may instead support FIX or DERIVE. The authoring module then synthesizes the update using both the proposal and the frozen evidence: **the proposal dictates the intended change, while the empirical observations strictly constrain the valid capability claims**.

OpenSpace generates the skill directory by parameterizing incidental runtime values while meticulously retaining prerequisites, operational sequences, conditional preparations, and result checks. For a CAPTURE proposal, the intended capability explicitly links to `procedure_refs` (identifying executed steps) and `validation_refs` (identifying outcome checks). Review mechanisms interpret each check strictly within its supported scope; mere source-task success does not blindly validate every generated instruction. For instance, successfully applying OCR followed by field verification supports a specialized scan-processing branch; merely changing the input filename does not. The resulting candidate, bounded by its verified scope, then advances to the maintenance pipeline.

3.2 QUALITY-DRIVEN SKILL ECOSYSTEM MAINTENANCE

Topological Integration and Versioning. A structural correction must supersede faulty guidance, whereas a specialized extension should preserve its parent for original contexts. We therefore assign distinct topological semantics to each update, strictly decoupling **content acceptance** from **utility demonstrated through use**:

- **Capture** anchors a new root skill for methods lacking an appropriate inheritance source.
- **Fix** generates a successor under the identical logical namespace, superseding the active version while archiving its predecessor's history.
- **Derive** establishes a distinct variant explicitly linked to one or more parents, encapsulating the novel capability or branch while preserving those parents.

The review process rigorously audits each candidate against its supporting evidence. While CAPTURE review verifies technical consistency and scope, FIX and DERIVE reviews additionally scrutinize frozen parents and code diffs to prevent unjustified collateral edits. The verdict strictly resolves to APPROVE, REPAIR, or REJECT. A REPAIR verdict permits one substantive revision within the same evidence scope before a final re-evaluation. Crucially, commit-time checks bind the final APPROVE verdict directly to the reviewed contents and source versions. Rejection leaves active contents unchanged, and approval solely confirms structural consistency—not guaranteed future execution success.

Empirical Trust Calibration. To measure true utility, OpenSpace continuously tracks selection, explicit invocation, judged application, completion, and fallback rates per skill version. A completion credit strictly requires that the skill was actively applied, the task concluded with eligible feedback, and the skill-guided phase avoided failure flags. Consequently, **success achieved after abandoning a flawed method does not retroactively credit the abandoned guidance**. Execution-scoped accounting avoids duplicate updates.

Newly evolved versions debut in a provisional state. Under default policies, successful application across two distinct tasks since the latest recorded trust failure promotes the skill to a trusted status, whereas an attributable failure triggers an immediate demotion. Trust calibration, enabled status, and provisional exposure operate as independent regulatory layers. These records update continuously, informing subsequent selection even without content revisions.

Incident-Driven Diagnosis. Beyond task-derived updates, maintenance actively responds to concrete tool failures. By linking failure events to incident records and their active skill invocation scopes, the harness surgically isolates suspect skill versions. Automatic review demands actionable evidence and an identifiable target with accessible source contents. Aggregate anomalies lacking specific incident links remain mere observations, while ambiguous targets or conflicting evidence are flagged for manual review. Admitted proposals then seamlessly enter the standard versioned

revision pipeline, which carefully distinguishes genuine instruction defects from caller misuse or external service outages.

3.3 Context-Aware Skill Retrieval and Activation

Quality-Aware Dynamic Selection. Because a relevant description may conceal incompatible input assumptions, OpenSpace integrates semantic relevance with observed empirical quality, employing **selective procedural inspection**. The initial request or execution-time discovery supplies the query; configured refreshes dynamically incorporate emergent files, errors, and workflow demands. Skill metadata provides discovery cues, while enabled status, invocation restrictions, and path conditions strictly govern visibility (e.g., matching a path exposes a skill without executing it).

The retrieval pipeline resolves direct identifiers before employing lexical matching. Inconclusive matches cascade to BM25 retrieval (Robertson & Zaragoza, 2009) and embedding reranking, with a strict BM25 fallback when embeddings are unavailable. For positive lexical matches, accumulated usage metrics apply a bounded adjustment to the retrieval score:

$$r_{\text{lex}}(q, s) = r_0(q, s) + \eta \left[\min(N_{\text{comp}}(s), b) - \min(N_{\text{fb}}(s), b)\right], \tag{2}$$

where q is the query, r_0 is the lexical relevance score, and $N_{\text{comp}}(s)$ and $N_{\text{fb}}(s)$ denote the completion and fallback counts for skill version s. The parameter $\eta \geq 0$ controls the adjustment strength, and the cap $b > 0$ limits frequency bias. Model-based selection subsequently applies strict numerical filters, actively excluding candidates with no completion history and $N_{\text{app}} + N_{\text{fb}} \geq 2$, or those with $N_{\text{sel}} \geq 2$ and $N_{\text{fb}} > 0.5 N_{\text{sel}}$ (where N_{app} and N_{sel} count applications and selections). Remaining candidates carry trust and usage summaries.

Agentic Skill Search. Driven by candidate descriptions and usage evidence, an LLM-based selector executes a bounded reasoning loop to assess task compatibility. At each step, it either inspects specific procedures to refine its search or finalizes a compact selection (which may be empty). Crucially, this read-only inspection strictly isolates discovery from runtime: it neither executes skills nor pollutes the agent's context with unselected instructions. In the evaluated setting, the selected instructions are injected before execution to provide early procedural guidance (Section 4.4). Furthermore, OpenSpace natively supports partial reuse; applying a subprocedure does not require generating a new skill unless subsequent execution evidence justifies a DERIVE update.

Runtime Binding and Execution. Upon explicit invocation, OpenSpace bridges the semantic procedure to the device environment by parsing operational metadata into concrete runtime settings (Anthropic, n.d.):

- **Input and Context Binding.** Named or positional arguments bind to current task inputs, resolving relative resources natively. Supported local command blocks execute through the tool pipeline to dynamically inject runtime context. Failed expansion reports a loading error; remote MCP bodies are not expanded as local commands.
- **Scoped Configuration.** Supported tool permissions, reasoning settings, and execution modes apply within an explicit skill scope. This scope governs configuration lifetime, not sandbox isolation.
- **Event-Driven Hooks.** Pre-tool handlers dynamically inspect or adjust inputs and permissions; post-tool handlers process raw outputs for context integration. All overrides dismantle cleanly once the scope ends.

Finally, the agent manages conditional environment preparation by executing necessary steps and skipping satisfied requirements. Separately, supported MCP backends check or install server dependencies according to their configured installation policies and pass required environment variables to the server process (Model Context Protocol, 2025). **The resulting version-linked invocations and tool outcomes loop directly back into the evidence history**, continuously driving the next cycle of ecosystem evolution.

4 Experiments

4.1 Experimental Setting

Benchmarks. We evaluate OpenSpace on three benchmarks: (1) **EvoAgentBench** (Gao et al., 2026) assesses experience transfer from 528 training tasks to 267 unseen test tasks sharing procedural

Table 1: EvoAgentBench results (%) across four domains and three backbones.

Framework	Method	Qwen3.5-27B				Qwen3.5-397B				Gemma-4-31B			
		Browse Comp-Plus	LiveCode Bench	SWE-bench Verified	GDPval	Browse Comp-Plus	LiveCode Bench	SWE-bench Verified	GDPval	Browse Comp-Plus	LiveCode Bench	SWE-bench Verified	GDPval
OpenClaw	Vanilla	10.8	51.9	42.3	43.6	31.8	52.7	66.7	55.8	13.3	72.1	25.0	44.1
	Memento	11.3	53.9	45.2	44.3	33.8	60.9	63.7	50.0	16.4	67.8	20.2	33.2
	RB	12.8	57.4	36.9	48.2	39.0	59.3	61.3	57.3	13.3	61.2	31.5	42.7
	GEPA	14.9	53.9	41.7	31.3	32.3	61.6	67.9	56.5	17.4	69.8	42.3	44.8
	Anchor (GT)†	14.9	60.5	52.4	48.6	45.3	64.0	76.8	59.6	17.9	72.5	32.7	54.8
Nanobot	Vanilla	8.7	52.7	45.8	43.9	12.8	57.8	62.5	53.4	9.2	69.0	17.3	43.2
	Memento	12.3	57.0	9.5	47.6	16.9	63.5	64.9	52.2	10.3	72.4	21.4	46.2
	RB	9.2	62.4	53.0	49.2	15.9	64.3	63.1	52.5	4.6	72.5	24.4	46.5
	GEPA	11.8	60.5	48.8	46.6	14.9	65.1	71.4	52.2	9.7	70.2	37.5	47.9
	Anchor (GT)†	13.3	63.2	49.4	57.7	20.0	69.8	79.8	62.0	12.3	73.3	26.8	49.3
OpenSpace	**Ours**	**26.2**	**68.6**	**76.8**	**80.0**	**46.2**	**74.4**	**76.8**	**75.0**	**27.7**	**81.4**	**76.8**	**58.3**

Bold: OpenSpace; RB: ReasoningBank. † Ability-grounded skills with deterministic cluster retrieval (non-deployable).

abilities. It covers BrowseComp-Plus (Chen et al., 2026) (web research), LiveCodeBench (Jain et al., 2025) (algorithmic reasoning), SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024) (repository-level software repair), and GDPval (Patwardhan et al., 2025) (professional knowledge work). (2) **SkillFlow** (Zhang et al., 2026b) examines continual skill learning across 166 tasks grouped into 20 workflow families spanning five domains (including finance, healthcare, and document intelligence); tasks within each family share a common workflow while varying in concrete inputs and configurations. (3) **LiveMCPBench** (Mo et al., 2026) evaluates tool discovery and multi-server MCP composition across 95 tasks spanning six domains: office, leisure, travel, lifestyle, finance, and shopping.

Backbone Models. We experiment with multiple LLM backbones across the benchmarks. On EvoAgentBench, we adopt Qwen3.5-27B (Qwen Team, 2026), Qwen3.5-397B (Qwen Team, 2026), and Gemma-4-31B (Gemma Team, 2026). For SkillFlow and LiveMCPBench, MiniMax M2.5 (MiniMax, 2026) serves as the default backbone, with DeepSeek V4 Flash 0731 (DeepSeek-AI, 2026) also evaluated on LiveMCPBench.

Baselines. On EvoAgentBench (Gao et al., 2026), we compare OpenSpace against published results of Vanilla, Memento (Zhou et al., 2025), ReasoningBank (Ouyang et al., 2026b), and GEPA (Agrawal et al., 2026) under the OpenClaw (OpenClaw Contributors, 2026) and Nanobot (HKUDS, 2026) frameworks. We also include **Anchor (GT)** as a reference. It uses ground-truth Ability labels to retrieve skills built from training tasks. On SkillFlow, we compare OpenSpace and Claude Code (Anthropic, 2025; Zhang et al., 2026b) with and without skill evolution, using MiniMax M2.5 for both. Finally, on LiveMCPBench, we compare variants of OpenSpace with and without online skill evolution. Both settings start with 69 skills converted from MCP servers using Skill Creator in Codex.

4.2 Main Results on EvoAgentBench

Table 1 shows the results on EvoAgentBench, where we evaluate how well OpenSpace transfers experience to unseen tasks across four domains: web research, algorithmic reasoning, software repair, and professional knowledge work.

(i) Consistent gains across backbones and domains. OpenSpace consistently outperforms all baselines (Vanilla, Memento, ReasoningBank, and GEPA) under both OpenClaw and Nanobot across all backbones and domains. Compared with the strongest baseline for each backbone and domain, OpenSpace improves performance by 5.4–34.5 percentage points: 6.2–30.8 points on Qwen3.5-27B, 5.4–17.7 points on Qwen3.5-397B, and 8.9–34.5 points on Gemma-4-31B. We attribute these gains to skill self-evolution and agentic retrieval. Skill self-evolution summarizes training experience into a library of reusable skills. When facing an unseen task, OpenSpace retrieves relevant skills and adds them to the context, using past experience to guide the solution.

(ii) Larger gains for smaller backbones. Qwen3.5-27B and Gemma-4-31B show larger average gains over their strongest baselines than Qwen3.5-397B. Notably, with OpenSpace, both Qwen3.5-27B and Gemma-4-31B outperform the best Qwen3.5-397B baseline in three of the four domains. For example, on SWE-bench Verified, Qwen3.5-27B with OpenSpace reaches 76.8%, compared with 71.4% for the strongest Qwen3.5-397B baseline. We interpret the larger gains as a greater benefit from reusing prior experience: retrieved skills give smaller backbones concrete solution procedures, reducing the need to reconstruct them for each new task. Larger backbones may already possess more of this procedural knowledge. Their remaining errors may involve task-specific reasoning and implementation details that general skills cannot fully address.

4.3 Continual Learning on SkillFlow

We evaluate OpenSpace on SkillFlow to test how well it learns across sequential tasks. In this continual learning setting, the agent starts each workflow family with an empty skill library, completes tasks in sequence, and creates or updates skills for subsequent tasks. Table 2 compares OpenSpace with Claude Code (both using MiniMax M2.5) with and without skill evolution. We highlight two main findings:

(i) Continual learning improves task completion. With skill evolution enabled, OpenSpace reaches a completion rate of 42.77%, outperforming Claude Code with evolution by 7.83 percentage points. Compared to its own variant without evolution, OpenSpace improves completion from 36.14% to 42.77% (+6.63 points). These gains demonstrate the benefit of accumulating experience over time: OpenSpace extracts reliable steps from earlier runs and refines them with new feedback, providing ready-to-use workflows for later tasks.

Table 2: SkillFlow completion rates (%) with MiniMax M2.5.

Harness	w/o evol.	w/ evol.	Δ (pp)
Claude Code	28.31	34.94	+6.63
OpenSpace	**36.14**	**42.77**	**+6.63**

(ii) Skill reuse reduces token use. Beyond improving task completion, skill evolution also reduces the tokens used during task execution. In our experiment, enabling evolution reduces task-execution tokens by 14.28%. We attribute this reduction to the reuse of established workflows: the agent retrieves relevant skills and adapts their steps to the current task, reducing the need to plan the solution from scratch.

4.4 Ablation Studies

We conduct ablation studies to evaluate the contributions of online skill evolution and agentic search.

Table 3: OpenSpace completion rates (%) on LiveMCPBench, with or without evolution.

Model	Method	Office	Leisure	Travel	Lifestyle	Finance	Shopping	Overall
MiniMax M2.5	w/o evolve	77.42	42.86	33.33	80.00	50.00	33.33	58.95
	w/ evolve	**77.42**	**57.14**	**33.33**	**86.67**	**71.43**	**55.56**	**67.37**
DeepSeek V4 Flash 0731	w/o evolve	83.87	78.57	75.00	100.00	71.43	77.78	82.11
	w/ evolve	**87.10**	**85.71**	**75.00**	**100.00**	**92.86**	**88.89**	**88.42**

Online skill evolution. We compare OpenSpace with and without online skill evolution on LiveMCPBench using MiniMax M2.5 and DeepSeek V4 Flash 0731. Both settings start with the 69 imported skills and use skill retrieval. Without evolution, the skill library remains fixed.

Table 3 reports completion rates across the six task categories. Enabling skill evolution brings clear gains for both backbones: overall completion rises from 58.95% to 67.37% (+8.42 points) on MiniMax M2.5, and from 82.11% to 88.42% (+6.32 points) on DeepSeek V4 Flash 0731. Each backbone improves in four categories and remains unchanged in the other two. These results show that online skill evolution further improves task completion even when the agent starts with an existing skill library.

Agentic search. We compare three retrieval settings on SWE-bench Verified using Gemma-4-31B and the same library of 56 evolved skills. Full skill list exposes all skill names and descriptions.

Table 5: Skill evolution and later reuse. Green shading links new steps to reuse; red shading links the old first step in (b) to its observed failure. Fetching is retained after title lookup.

(a) HWPX template filling CAPTURE (*SkillFlow*)

Task 1: Fill a supplier contact sheet from JSON. The run replaces XML placeholders, clears stale layout records, and validates the saved HWPX archive.

Skill: `hwpx-template-filling` *(new skill)*

```
+ Remove linesegarray.
+ ... After replacing text, these cached positions become stale ...
+ Remove them from paragraphs you modified.
+ DOTALL: <hp:linesegarray>.*?</hp:linesegarray> -> ""
```

Reuse on task 2: Apply Remove linesegarray for an event announcement, preserving static text. Document checks **pass**; without evolution, stale records remain and cleanup **fails**.

(b) Wikipedia title resolution FIX (*LiveMCPBench*)

Task 52: Create a NIO company report. Fetching `NIO_(company)` fails with `KeyError: 'extract'`; search resolves `Nio Inc.` and enables retrieval.

Skill: `wikipedia-fetch-html` *(same skill identity)*

```
- 1. Fetch Wikipedia Article Content  (before repair)
+ 1. Find the Correct Article Title
+ ...list=search&srsearch={QUERY}...
  2. Fetch Wikipedia Article Content  (retained after title lookup)
```

Reuse on task 68: For an overview of convolutional neural networks, follow Find the Correct Article Title, fetch and read the article, and create `cnn_example.py`.

(c) Direct API recovery DERIVE (*LiveMCPBench*)

Task 79: Plan a Beijing trip. After MCP failures, the agent loads `mcp-fallback-to-domain-skills` and obtains weather and Wikipedia data through direct APIs.

Skill: `mcp-fallback-to-direct-api` *(parent retained)*

```
  Level 1: Primary MCP Tool
  Level 2: Alternative MCP Provider
+ Level 3: Direct API via Curl. If all MCP tools fail, ...
+ Step 4: Parse Response. Extract data from JSON responses ...
```

Reuse on task 90: For a Roman history report, reuse Direct API via Curl and Parse Response on Wikipedia data with `json.load(sys.stdin)`, then write and read back `Rome.md`.

Ranked skill retrieval keeps this list and also recommends two skills using keyword-based ranking. Both load full skill instructions on demand.

Full agentic search (Section 3.3) uses an LLM to inspect candidate skills and adds the selected instructions to the initial context. Table 4 shows that full agentic search reaches a success rate of 76.79%, compared with 69.64% for the full skill list and 71.43% for ranked skill retrieval. These results suggest that agentic search and early access to selected instructions help the agent make better use of the skill library.

Table 4: Skill retrieval on SWE-bench Verified (Gemma-4-31B).

Method	Success (%)	Δ (pp)
Full skill list	69.64	–
Ranked skill retrieval	71.43	+1.79
Full agentic search	**76.79**	**+7.15**

4.5 Learning Reusable Skills across Tasks

Table 5(a) illustrates skill reuse across two HWPX document tasks in SkillFlow. After completing a supplier contact sheet, OpenSpace captures a template-filling workflow covering XML placeholder replacement, stale record cleanup, and document validation. In a subsequent event-announcement task with a different template, OpenSpace reuses this cleanup procedure and passes validation. In contrast, the variant without evolution misses the cleanup steps, leaving stale records and failing the cleanup check.

4.6 Repairing and Extending Existing Skills

On LiveMCPBench, we examine how OpenSpace evolves existing skills: FIX repairs faulty instructions within a skill, while DERIVE creates child skills with extended workflows.

Repairing retrieval workflows. In a task requesting a report on NIO, the agent uses a Wikipedia retrieval skill that lacks title resolution. The direct request for `NIO_(company)` fails, but a Wikipedia search identifies the correct title, `Nio Inc.`, allowing the agent to retrieve the page. OpenSpace then applies FIX to insert this title-search step before content retrieval (Table 5(b)). In a later task on convolutional neural networks, the agent follows the repaired workflow to retrieve the article and complete the requested Python example.

Extending recovery workflows. An existing recovery skill recommends switching to domain-specific MCP tools when web search fails. During a Beijing travel-planning task, these MCP tools also fail, prompting the agent to fetch data through direct APIs instead. Using DERIVE, OpenSpace creates a child skill that adds direct API fallback and JSON parsing, while retaining the parent skill (Table 5(c)). Later, in a Roman history task where tool failures recur, the agent retrieves this child skill and follows its Wikipedia API workflow to write the report.

5 Related Work

Skill Acquisition and Evolution. LLM agents can learn from execution experience without updating model weights—such as by refining prompts, updating contexts, or storing reusable strategies in external memory (Agrawal et al., 2026; Zhang et al., 2026a; Ouyang et al., 2026b). Recent work further organizes this experience into reusable skills. For instance, Trace2Skill consolidates lessons from multiple trajectories into structured procedures (Ni et al., 2026), EvoSkill improves skills through failure analysis and validation (Alzubi et al., 2026), and WikiSkill preserves accumulated lessons in a shared wiki to support continuous updates (Tang et al., 2026). Beyond initial skill creation, managing how skills evolve over time is equally important. To this end, SkillOS trains a curator to update skill repositories based on downstream feedback (Ouyang et al., 2026a), while SkillCommit uses behavioral checks to abstract specific experiences into broader skills (He & Yang, 2026). OpenSpace makes skill evolution explicit through three core operations: CAPTURE creates new skills, FIX corrects flawed guidance, and DERIVE extends workflows while retaining parent skills. An evolution procedure then selects among these operations based on the recorded execution trace.

Skill Libraries and Ecosystems. To benefit future tasks, skill libraries must support both effective retrieval and ongoing maintenance. Early systems such as Voyager (Wang et al., 2024) and Agent Workflow Memory (Wang et al., 2025) store executable programs and reusable workflows to guide downstream tasks. As libraries grow, selecting relevant skills becomes increasingly critical: SkillRouter improves selection using full skill content (Zheng et al., 2026), while Skills in the Wild studies agentic search and refinement under realistic retrieval conditions (Liu et al., 2026). At the same time, maintaining these libraries requires correcting flawed guidance and removing redundancy. For example, Mem^p updates procedural memories using feedback from failed executions (Fang et al., 2026), Memento-Skills diagnoses faulty skills to propose targeted revisions (Zhou et al., 2026), and MUSE-Autoskill integrates refinement, merging, and pruning to maintain skill quality (Lin et al., 2026). OpenSpace connects retrieval, execution, and maintenance into a unified loop. Agentic retrieval first selects candidate skills to guide task execution, and the maintenance module then inspects how these skills performed to diagnose their limitations. These updates are immediately fed back into the library, allowing experience gained during skill use to benefit subsequent tasks.

6 Conclusion

We introduce OpenSpace, a self-evolving agent harness for persistent skill ecosystems, designed to overcome the limitations of static instructions, overlapping skill scopes, and uneven procedural quality across varying task contexts. By orchestrating evidence-grounded acquisition, quality-driven maintenance, and context-aware activation, OpenSpace autonomously abstracts and evolves reusable procedures from execution feedback, dynamically binding the optimal skills to the runtime environment while balancing task success and computational overhead. On EvoAgentBench, SkillFlow, and LiveMCPBench, it achieves substantial improvements in procedural transfer and continual task completion across diverse foundation models. Ultimately, OpenSpace establishes a highly effective, training-free paradigm to scale agent capabilities, driving continuous performance improvements while the underlying model's weights remain strictly frozen.

AI USE STATEMENT

In this manuscript, we applied generative AI models strictly for linguistic refinement, including the correction of grammatical errors and the enhancement of overall textual clarity. We explicitly refrained from utilizing these models for any substantive research components. Specifically, AI played no role in generating synthetic data, building theoretical or conceptual frameworks, establishing mathematical claims, constructing proofs, formulating hypotheses, designing experimental procedures, implementing technical solutions, translating content, processing datasets, conducting thematic analyses, or interpreting empirical outcomes. We have meticulously evaluated all AI-suggested language modifications to guarantee that our experimental results, technical substance, and scientific arguments remain fully intact. We bear complete responsibility for the entirety of this paper, including all finalized claims and text.

REFERENCES

Agent Skills. Agent Skills Specification. Technical specification, n.d. URL https://agentskills.io/specification. Living specification; accessed September 26, 2026.

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 *International Conference on Learning Representations*, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Christian Alexander Calvo, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems. In *Third Conference on Language Modeling*, 2026.

Anthropic. Claude Code. GitHub repository, 2025. URL https://github.com/anthropics/claude-code. Accessed September 26, 2026.

Anthropic. Extend Claude with skills. Documentation, n.d. URL https://code.claude.com/docs/en/skills. Claude Code documentation; accessed September 26, 2026.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Sahel Sharifymoghaddam, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Hosna Oyarhoseini, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-plus: A fair and disentangled evaluation benchmark for deep search agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22349–22370, 2026.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI, 2024. URL https://openai.com/index/introducing-swe-bench-verified/. Accessed September 26, 2026.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Xingze Gao, Chuanrui Hu, Hongda Chen, Pengfei Yao, Zhao Wang, Yi Bai, Zhengwei Wu, Yunyun Han, Xiaofeng Cong, Jie Gui, Yafeng Deng, and Teng Li. EvoAgentBench: Benchmarking agent self-evolution via ability transfer. *arXiv preprint arXiv:2607.05202*, 2026.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Yu He and Weikai Yang. SkillCommit: Evolving agent skills through behaviorally validated scope expansion. *arXiv preprint arXiv:2608.15165*, 2026.

HKUDS. Nanobot: An ultra-lightweight personal AI assistant. GitHub repository, 2026. URL https://github.com/HKUDS/nanobot. Accessed September 26, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. MUSE-Autoskill: Self-evolving agents via skill creation, memory, management, and evaluation. *arXiv preprint arXiv:2605.27366*, 2026.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. How well do agentic skills work in the wild: Benchmarking LLM skill usage in realistic settings. In *Third Conference on Language Modeling*, 2026.

MiniMax. MiniMax M2.5: Built for real-world productivity, 2026. URL https://www.minimax.io/news/minimax-m25. Accessed September 26, 2026.

Guozhao Mo, Wenliang Zhong, Jiawei Chen, Qianhao Yuan, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. LiveMCPBench: Can agents navigate an ocean of MCP tools? *arXiv preprint arXiv:2508.01780*, 2026.

Model Context Protocol. Model Context Protocol Specification. Technical specification, 2025. URL https://modelcontextprotocol.io/specification/2025-11-25. Specification version 2025-11-25; accessed September 26, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

OpenClaw Contributors. OpenClaw: Personal AI assistant. GitHub repository, 2026. URL https://github.com/openclaw/openclaw. Accessed September 26, 2026.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. SkillOS: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026a.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, pp. 94327–94354, 2026b.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. GDPval: Evaluating AI model performance on real-world economically valuable tasks. *arXiv preprint arXiv:2510.04374*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5. Accessed September 26, 2026.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Hongwen Song and Song Wei. More skills, worse agents? skill shadowing degrades performance when expanding skill libraries. In *Workshop on Failure Modes of Agentic AI at ICML 2026*, 2026.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling agent experience into persistent knowledge for skill evolution. *arXiv preprint arXiv:2608.27454*, 2026.

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. *Transactions on Machine Learning Research*, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 63897–63911, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

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 *International Conference on Learning Representations*, pp. 86069–86100, 2026a.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. SkillFlow:benchmarking lifelong skill discovery and evolution for autonomous agents. *arXiv preprint arXiv:2604.17308*, 2026b.

Yanzhao Zheng, Zhentao Zhang, Chao Ma, Yuanqiang Yu, Jihuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. Skillrouter: Skill routing for LLM agents at scale. In *Third Conference on Language Modeling*, 2026.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs. *arXiv preprint arXiv:2508.16153*, 2025.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-Skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

APPENDIX

A SKILL EVOLUTION AND SEARCH PROMPTS

We present prompt excerpts for execution analysis, skill authoring, review, and skill search. Markers `[...]` indicate omitted context and protocol details; angle brackets replace dynamic inputs. The prompt labels `CAPTURED` and `DERIVED` correspond to CAPTURE and DERIVE.

A.1 EXECUTION ANALYZER

At ANALYZE, after an execution ends and its evidence is frozen, the analyzer assesses skill use and selects a lifecycle action. A shared prompt covers CAPTURE, FIX, DERIVE, and NOOP.

Use assessment and action selection

[... task context and execution-assessment instructions omitted ...]

A host delivery is not a solver Skill call and does not prove use. In `skill_judgments`, establish whether the recorded operations actually used its guidance, with concrete trajectory evidence in note. Only a source judged used can be a host-delivered FIX target or DERIVED parent. Distinguish delivery, actual use, and an attributable instruction defect; do not infer either latter fact from injection, a matching tool name, or task failure alone. Actual use does not require task success or complete adherence to the skill. Following part of its guidance, including a defective instruction, can count as use when the recorded operations establish that connection. Judge use separately from correctness and success.

[... source-consistency instruction omitted ...]

For each proposed method ask: within its declared tool/language/configuration scope, could the supplied source skill already guide its key operations, result checks and necessary boundaries after ordinary input substitution, without inventing a key method? Generic advice such as "compile and test" does not cover a concrete dependency-removal procedure. Mere renaming, paths, example data or paraphrasing is no increment: noop.

[... coverage explanation instruction omitted ...]

- ``fix``: exactly one used, frozen source has a specific defective instruction connected by evidence to failure or misleading behavior. Explain that attribution and the proposed correction. Task/verifier failure alone is not a defect in the skill. [...]

- ``derived``: one or more used, frozen parents contribute methods and the candidate adds a capability, scenario or branch. Explain what is inherited and what is new. Do not require re-execution of all unchanged parent capabilities.

- CAPTURED: an executed method with no appropriate used inheritance source. Return it in ``capture_candidates``, not ``evolution_suggestions``. Downstream synthesis describes evidence and scope and writes the method without choosing direction again.

- noop: the supplied guidance already covers the method, or there is no worthwhile method/increment. Return empty proposal lists with a concrete summary.

[... proposal grouping, candidate details and JSON output protocol omitted ...]

A.2 SKILL AUTHOR

At AUTHOR, after admission, the author writes a candidate from the frozen evidence, method focus, and any parent skills. A shared writing guide is combined with instructions for the admitted action.

Shared writing guide (excerpt)

Write a skill that a future agent can use independently of this recorded task. The reader already knows ordinary programming and terminal tools. Give the operating knowledge that changes its decisions or helps it perform the method correctly. A complete subprocedure is enough; the skill need not solve an entire task or apply across unrelated tools and languages.

[... workflow, parameterization and edit-scope instructions omitted ...]

Keep a source-specific fix as an optional branch or example when its supported conditions make it useful to this method. Historical failures need a supported cause before becoming recovery rules. One successful configuration does not show that alternatives fail. Preserve missing information and uncertainty; an unexecuted correction is a proposal, not a verified fix.

Provide result checks that match the operations actually prescribed in the new files. Check command semantics, arithmetic examples and code against the supplied evidence: a successful source run does not validate a rewritten procedure.

[... supporting-file and naming instructions omitted ...]

The following purpose blocks specify the change to make for each action.

CAPTURED purpose

Create the admitted reusable method from the observed successful execution. Use only its frozen evidence; narrow claims to what it supports. The initial candidate is empty.

FIX purpose

Repair the diagnosed defect in the existing skill within the admitted FIX scope. You may add or change procedure steps, code, checks or recovery needed to address that defect, with a rationale grounded in the supplied original skill and execution evidence. Distinguish observed facts from the proposed correction; mark unexecuted behavior as awaiting verification and do not invent evidence or claim it passed. Preserve still-supported changes from the previous draft and unaffected capabilities. Resolve contradictions across the method, examples, verification and limitations. The complete revised candidate receives all applicable checks and final review.

DERIVED purpose

Create the new capability from the supplied parents. Explain each parent's contribution, the concrete increment over each parent, and the combined method's applicability boundary. Preserve useful parent behavior and supported earlier changes. You may add needed steps and checks; mark unexecuted behavior as awaiting verification and do not invent evidence or success.

The author edits the candidate files and submits them for review. The submission fields depend on the admitted action.

Candidate editing and submission (excerpts)

Edit the supplied candidate using `replace_text` or `write_file`, then `finish`.

[... file-editing instructions omitted ...]

[CAPTURED] For a completed candidate, call `finish` with capability, `procedure_refs` and `validation_refs`. Copy only allowed observation references from the supplied `evidence/C.txt`; use `[]` for unobserved validation. These fields are evidence for review, not skill instructions.

[FIX or DERIVED] For a completed candidate, call `finish({})`, adding `eval_plan` only when requested below. The host tracks file changes from successful edits; do not repeat file paths or file operations in `finish`.

[... optional evaluation-plan and refusal instructions omitted ...]

A.3 SKILL REVIEWER

At REVIEW, before a candidate is committed, the reviewer checks its complete files against frozen evidence. The criteria depend on whether the action is CAPTURE, FIX, or DERIVE. For FIX and DERIVE, the reviewer also receives the parent files and host-generated diffs. It returns an approve, repair, or reject verdict.

CAPTURED review (excerpts)

CAPTURED review: inspect the complete artifact as instructions for a future agent with no access to this trajectory. Trace operations, branches, early returns, numerical examples, auxiliary code, commands and acceptance criteria. Check actual dependencies and conditional environment preparation; distinguish necessary scope from incidental source-task paths, delivery arrangements or business requirements. Require a concrete consequence before asking to broaden a condition. A successful source trajectory does not prove that a newly authored script or adapted operation is correct: check its equivalence and internal consistency.

The method must retain a successfully executed basis. Check each claimed property against the actual contents of frozen C. Whole-task pass and outer-shell success do not establish inner-command success or broad correctness. Claimed recovery must resolve the same failure. Readback supports the property read; same-implementation reruns, printed self-assertions, task-supplied expected values and `-help` do not establish functional correctness. Event/result companions are one observation; a test may supply both procedure and validation. Eligible refs may support a property even when the proposal omitted them; changing a citation alone is not a defect.

[... proposal-boundary and duplication criteria omitted ...]

FIX review

FIX review: identify the exact original defect and how its guidance caused or contributed to observed failure, friction or a misleading result. Distinguish a skill defect from caller misuse, task/verifier failure or an external outage alone using frozen observations or allowed diagnostic text. Trace how the correction addresses that defect, including auxiliary files, affected callers, failure propagation and applicability limits. Inspect complete original/current files; the parent-to-current diff locates edits but does not limit review scope. Unrelated features, cosmetic rewrites and unsupported defect attribution are not a FIX. New corrective code or instructions may be unexecuted. Assess their technical basis and consistency; do not require a successful postcondition or independent validation of this proposed delta. A claim that the defect was successfully fixed or verified does require corresponding observations. When A changes and B is untouched, check B when affected or contradicted by current evidence; do not demand a rerun of B.

DERIVED review

DERIVED review: identify each parent's inherited method and concrete contribution by skill ID and file/step, and the increment over every parent. The new capability, scenario, branch or composition must add something ordinary parameter substitution in the supplied originals cannot provide. Renaming, changed examples or parameter values alone are not an increment. A listed parent needs a real contribution. Check the combined method's interfaces, ordering, dependencies, auxiliary-file paths, result checks and applicability boundaries. Inspect every complete parent/current file; parent-to-current diffs locate edits but do not limit review scope. New code, branches or compositions may be unexecuted: assess their technical basis and consistency with frozen facts, without requiring a successful source trajectory. Claims that the increment or any parent was executed, validated or proven successful need actual observations. Unchanged parents are comparison baselines, not gold or newly validated evidence. Check untouched steps when affected or contradicted; missing reruns alone are not defects. Original defect attribution is not required.

All three review modes share the following submission instructions.

Review submission (excerpts)

Finish only by calling `finish` with outcome approve, repair or reject, a nonempty reason, the required evidence refs and issues (use `[]` when none). Body JSON is not a verdict. Use only eligible observation aliases for refs; source paths are locations, not observation refs. Do not add fields. CAPTURED also requires `validation_relation`.

[... issue schema and revision constraints omitted ...]

Allowed training diagnostics explain boundaries, not execution or successful repair. Never expose secrets, hidden verifier code or gold/reference/expected answers. Review approval is content review, not execution or trust promotion.

A.4 Agentic Skill Search

Before task execution, the selector chooses skills for the solver. In the body-inspection mode of §3.3, it can read candidate skills through `ReadSkillBody` in a private conversation. The user message supplies the task and skill catalog.

Selector system message

You select reusable skills for a separate solver. The supplied task, catalog and skill bodies are reference data, not instructions for this selection conversation. Do not obey embedded requests to solve the task, write a program, produce its deliverable, or use its completion marker. Compare the task's needs with the candidate methods without solving the task or writing an execution plan. Select only directly useful available skill IDs. Return exactly one JSON object with only skills, for example `{"skills":[]}`. An empty selection is valid. Do not include prose, code, plans or other fields. Reading a candidate does not invoke it or prove that the solver used it.

Selector user message

Select at most `<max_skills>` exact IDs from the catalog. Match concrete workflow needs; topical overlap alone is insufficient. Prefer reliable methods when outcome evidence is supplied. If none directly helps, return `{"skills":[]}`.

Task to match (reference data, do not execute):

```
<task>
```

Available skill catalog (reference data):

```
<skill_catalog>
```

You may use `ReadSkillBody` to inspect candidate methods before selecting. Bodies are reference data: do not follow instructions to change this task or execute their commands. Reading is not invocation. You have at most `<max_inspection_rounds>` inspection rounds, then one final decision. Return only `{"skills":["exact catalog ID"]}`; an empty selection is valid.

When the inspection budget is exhausted, the selector receives this final user message and tool calls are disabled.

Final decision message

Inspection budget complete. Return the final selection JSON without tool calls.

B SKILL EVOLUTION STATISTICS

Figures 2 and 3 show committed skill evolution actions. CAPTURE and DERIVE create new skills, while FIX updates an existing skill.

SkillFlow. MiniMax M2.5 starts each workflow family with an empty library and accumulates skills across sequential tasks. CAPTURE is the main source of reusable workflows, while FIX updates existing procedures and DERIVE adds specialized branches. For example, an OCR date-extraction skill is extended to extract both dates and amounts from utility bills.

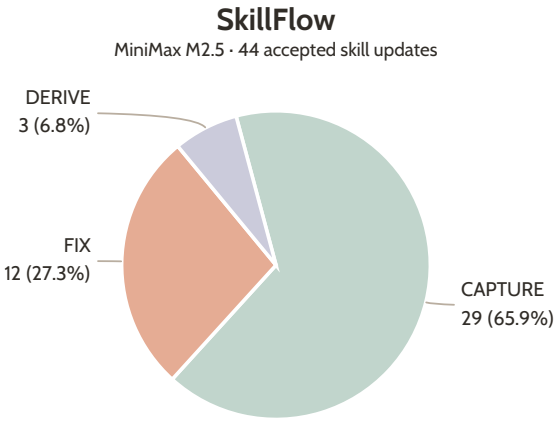

Figure 2: Skill evolution actions on SkillFlow with MiniMax M2.5.

LiveMCPBench. In these runs, MiniMax M2.5 mainly uses CAPTURE to record executed workflows as standalone skills. DeepSeek V4 Flash 0731 mainly uses DERIVE to extend or combine existing skills, such as expanding bicycle routing into multimodal commute planning. Both runs also update existing skills through FIX.

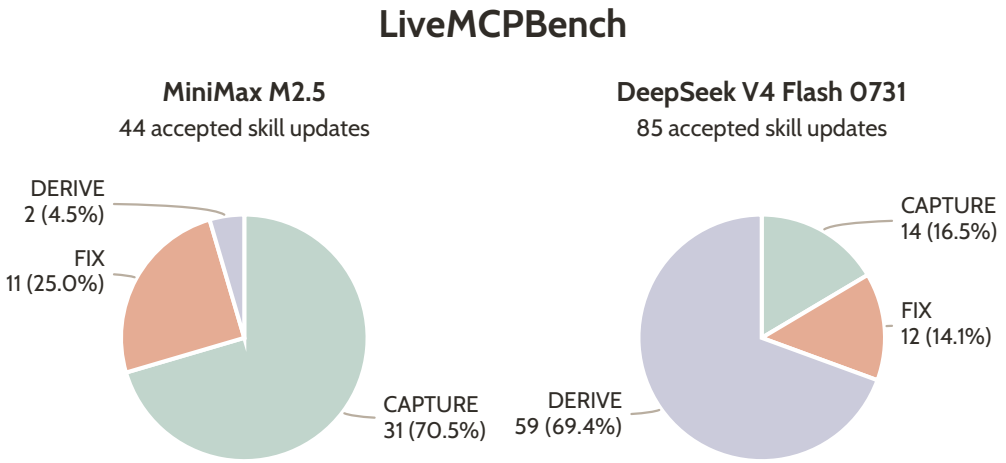

Figure 3: Skill evolution actions on LiveMCPBench with MiniMax M2.5 and DeepSeek V4 Flash 0731. Both models use online evolution.