SKILLEVOLVER: SKILL LEARNING AS A META-SKILL

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills, folders of instructions and scripts that an agent loads at inference time, are today *static artifacts*: authored once, by human curation or by one-shot generation from the model's own knowledge, and never improved by use. Methods that do learn skills from execution traces learn them from many instances of a task distribution. We study *online skill learning*, in which a single new task arrives and only a handful of trials can be afforded before its skill must ship. We propose **SkillEvolver**, a meta-skill, itself loaded like any other skill, that drives any CLI-agent to author, deploy and refine a skill for that task; it learns the skill's prose and code rather than model weights, so the result drops into any agent without retraining. Two design choices make a handful of trials enough: each trial follows a distinct strategy the agent writes itself, so that a small batch explores different approaches rather than resampling one, and, when a first batch leaves failures, the distilled skill is refined from how fresh agents fail while using it. On SkillsBench, skills learned by SkillEvolver outperform the human-curated skills in overall resolution rate on all three models we evaluate, raising the resolution rate to 57.9% on DeepSeek-V4-Flash (13.3 points absolute improvement), 62.2% on GLM-5.2 (7.2 points absolute improvement) and 65.5% on Claude Opus 4.6 (10.8 points absolute improvement). Even when all four trials of a first batch fail, a second batch solves the training task 64.4% of the time after distillation, against 29.7% when drawn blind, and the loop spends 6.57 exploration rollouts per task on average.

1 INTRODUCTION

Modern LLM agents increasingly require *procedural* knowledge to tackle complex real-world problems: not merely what a task is about, but how it should be carried out in a specific environment. For example, how to fill a domain-specific spreadsheet template, follow a project-specific schema, invoke a brittle tool interface, or avoid known failure modes in a recurring workflow. Agent skills have emerged as a lightweight mechanism for encoding such knowledge. Specifically, a skill is a short, task-specific artifact loaded at inference time, typically bundling natural-language instructions, executable scripts, reference files, examples, and usage constraints into a reusable dependency for an agent (Anthropic, 2025a;b; Xu & Yan, 2026). Unlike one-off prompts or demonstrations, skills can be stored, transferred, revised and redeployed, thus turning local procedural know-how into a portable unit of agent behavior. Human-curated skills have been shown to substantially improve agents' performance on skill-focused benchmarks, such as SkillsBench (Li et al., 2026b), and ecosystem-scale studies report hundreds of thousands of community-authored skills in circulation (Li et al., 2026a). Yet this skill-authoring paradigm still relies heavily on human expertise. In long-tail deployment settings, where new domain tasks arise on demand, recruiting a specialist to author for every workflow is neither timely nor economical. This raises the central question: *can an agent acquire procedural knowledge from a bounded set of deployment-time trials and package it as a reusable skill, without retraining model weights?*

Recent efforts to automate skill creation mainly fall into two directions, but neither assumes the on-demand, few-trial skill-authoring regime we target. *Parametric self-generation*, in which the model writes a skill directly from its pre-trained knowledge, as in the self-generation condition of SkillsBench (Li et al., 2026b), commits without grounded feedback, and this one-shot variant can perform no better than, and sometimes worse than, using no skill at all (Li et al., 2026b). Anthropic's official skill-creator (Anthropic, 2025c) adds a test-and-revise loop, but one steered by a human who

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.

states the intent and reviews the outputs. *Trace- and RL-based skill acquisition* (RL, Reinforcement Learning), by contrast, derives its strength from broad execution coverage: Trace2Skill (Ni et al., 2026) mines a pre-collected pool per domain (200 trajectories on its spreadsheet benchmark) through a multi-stage trace-distillation pipeline over a pre-collected trajectory pool; SkillRL (Xia et al., 2026) grows a recursively expanding skill library by pooling experiences across many training tasks; and a broader family of RL-based acquisition methods build on similar cross-task aggregation (Xu & Yan, 2026). These solutions are powerful, but they assume substantial offline preparation per domain. Many real-world tasks arrive on demand, one at a time, with only a handful of exploration trials affordable before a skill must ship. Our focus is therefore not ecosystem-scale management or large-pool offline consolidation (Li et al., 2026a), but *on-demand domain-specific skill learning*: how to author, deploy, audit and refine a reusable skill within the bounded experience of a single newly-arrived task.

In this work, we propose **SkillEvolver**, a lightweight solution framework for *online skill learning*: adapting an external skill artifact for a newly arrived task, rather than updating model parameters. This mirrors the timing of test-time adaptation (Sun et al., 2020; Wang et al., 2021), but SkillEvolver updates a reusable skill by observing a small number of training-time task trials and redeploying the revised artifact to fresh downstream agents. SkillEvolver uses a single *meta-skill* to drive a standard command-line interface (CLI)-agent through the lifecycle of skill acquisition: exploring a small number of trials on the task's available training split, authoring a reusable *domain skill*, deploying the candidate skill to fresh Domain-Skill Agents, and refining the artifact from their observed behavior (§3). Like recent skill learners, SkillEvolver grounds refinement in this deployment handoff rather than in self-reflection, since a candidate can also fail by omitting a needed instruction, exposing a misleading procedure, or being silently bypassed. What distinguishes our approach is its regime: a single new task and a handful of trials, where strategies the agent writes supply the contrast other methods draw from a pool of instances, and the agent decides when to stop (§2). A fresh-session Auditor checks each candidate for leakage, overfitting, and deployment-specific failures. This design turns a bounded set of training-time trials on a newly arrived task into a reusable procedural artifact for future agents.

Our contributions are summarized as follows:

- We formulate online skill learning as the acquisition of reusable procedural artifacts, and introduce SkillEvolver, a plug-and-play framework (meta-skill) that lets a standard CLI-agent author domain-specific skills without retraining model weights (§3).
- We bring deployment-grounded refinement, which tests each candidate skill as a dependency used by fresh agents, to the single-task, few-trial regime by pairing it with strategy-diversified trials that manufacture the contrast a single instance cannot supply, so that SkillEvolver exposes skill failures which are not visible from exploratory traces alone (§3.2.1–3.2.3).
- Experiments on SkillsBench (Li et al., 2026b), 87 tasks across 8 professional domains, show that skills learned by SkillEvolver outperform the human-curated skills in overall resolution rate on all three models we evaluate, by 13.3, 7.2 and 10.8 points on DeepSeek-V4-Flash, GLM-5.2 and Claude Opus 4.6 (§4.2). Ablations identify strategy diversification as the load-bearing component of the loop, worth 11.0 points (§4.3).

2 Related Work

Agent skills and their authoring. A skill is a task-specific directory of prose instructions, scripts and examples that an agent loads at inference time (Anthropic, 2025a; Xu & Yan, 2026). Ecosystem-scale work organises and benchmarks the community skills already in circulation (Li et al., 2026a). SkillsBench (Li et al., 2026b) pairs each task with a human-curated skill and a deterministic verifier, and finds that curation helps substantially while skills a model writes from its own parametric knowledge do not. Anthropic's skill-creator (Anthropic, 2025c) is itself a meta-skill: it drafts a skill, runs it in fresh subagents on a few prompts, and revises it under human review. SkillEvolver shares the meta-skill form and the fresh-agent test, but replaces the human with verifier-scored trials on a separate training variant of the task.

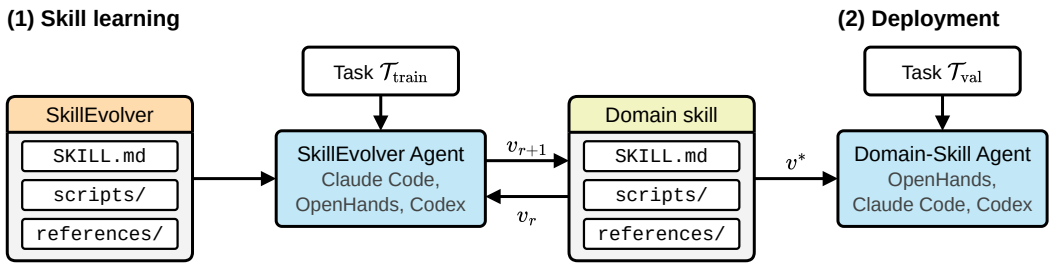

Figure 1: **SkillEvolver as a portable meta-skill.** (1) SkillEvolver is itself a skill: any CLI-agent that loads skills (e.g., Claude Code, OpenHands or Codex) loads it through the same interface as any domain skill and becomes the SkillEvolver Agent. Given the training split of a new task $\mathcal{T}=(\mathcal{T}_{\text{train}}, \mathcal{T}_{\text{val}})$, it repeatedly reads the current domain skill v_r and writes a revision v_{r+1} (Figure 2). (2) The final skill v^* is loaded by a fresh Domain-Skill Agent (OpenHands or Claude Code in our experiments) on the held-out split $\mathcal{T}_{\text{val}}$. The learned object has the same form as the meta-skill, prose instructions in `SKILL.md` plus scripts and references, rather than updated model weights.

Evolving context and memory. Persistent-memory agents carry state across sessions (Packer et al., 2023; Park et al., 2023). Dynamic Cheatsheet (Suzgun et al., 2026) and ACE (Zhang et al., 2026) grow a self-curated memory or playbook through incremental edits over a stream of queries or a training split. ReasoningBank (Ouyang et al., 2026) distils strategies from successes and pitfalls from failures, as judged by the agent itself, and retrieves them for later queries; its parallel mode contrasts several sampled trajectories of one query, the nearest analogue to our exploration batch. These methods work well where related queries recur, and what they learn is text held in or retrieved into the agent's context. Our skill is instead a directory, with executable scripts, that a separate agent must discover, invoke and run.

Skill acquisition from execution traces. A wave of concurrent work learns skills from recorded agent behaviour. SkillRL (Xia et al., 2026) distils a skill bank with a teacher model while training the policy's weights with reinforcement learning. SkillClaw (Ma et al., 2026b) evolves shared skills from the sessions of many users through an agentic evolver. Trace2Skill (Ni et al., 2026) turns each trajectory of a labelled pool into a patch and merges the patches hierarchically, and XSkill (Jiang et al., 2026) builds a skill library and an experience bank from sibling training tasks. SkillGen (Ma et al., 2026a) contrasts failures with successful neighbours and refines the skill over several verification rounds. EvoSkill (Alzubi et al., 2026) is closest to our loop: a proposer edits skills from the failures of an executor that holds them, and a labelled validation split decides which candidates survive. Contrasting successes with failures, and refining a skill from the failures of an agent that uses it, are therefore established, and we claim neither as new. The evidence these methods learn from, however, comes from many distinct instances: a labelled pool or split, sibling tasks, or a fleet of users. SkillEvolver learns from one task and a single generated training variant with its verifier, spending 6.57 exploration rollouts on average.

Trial diversity and reflection. Reflexion and Self-Refine (Shinn et al., 2023; Madaan et al., 2023) turn a failed attempt into verbal feedback for the next one, and ETO (Song et al., 2024) learns from paired failed and successful trajectories. In the skill learners above, the contrast between trials comes from the breadth of the instance pool or from sampling temperature. With one instance there is no pool, and temperature tends to vary the surface form of a trial more than its method (§4.3). SkillEvolver instead has the agent write a distinct strategy for each trial, so that a small batch covers different approaches and a failure can be traced to a stated design decision. Removing these strategies costs more than any other component in our ablations.

Agent scaffolds and evaluation. Skills build on scaffolds that interleave reasoning with action (Yao et al., 2023), call tools (Schick et al., 2023), act through code (Wang et al., 2024), coordinate several agents (Hong et al., 2024; Wu et al., 2023), and grow self-verified skill libraries (Wang et al., 2023). Agent benchmarks span code, web, operating-system and enterprise work (Jimenez et al., 2024; Zhou et al., 2024; Liu et al., 2024; Xie et al., 2024; Xu et al., 2025; Mialon et al., 2024). Contamination is a recurring concern in LLM evaluation (Sainz et al., 2023), and skill au-

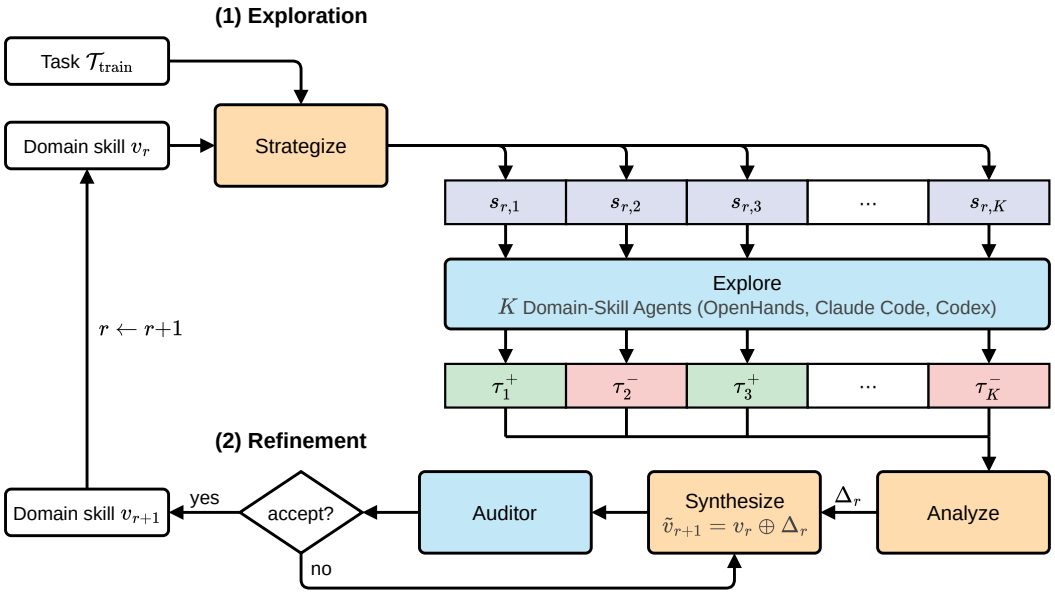

Figure 2: **One iteration of the SkillEvolver loop.** At iterations $r = 0, \ldots, R-1$, SkillEvolver observes only $\mathcal{T}_{\text{train}}$. (1) Strategize writes K diverse strategies $s_{r,i}$ (at r=0 after first reading the task); in Explore, K fresh Domain-Skill Agents each load v_r with one strategy and attempt $\mathcal{T}_{\text{train}}$, yielding successful ($\tau^+$) and failed ($\tau^-$) trajectories. (2) Analyze contrasts them into the missing guidance Δ_r, Synthesize patches v_r into the candidate $\tilde{v}_{r+1}$, and the Auditor checks it in a fresh session. An accepted candidate becomes v_{r+1}, the next iteration's input; a rejected one returns its violations to Synthesize as the next patch target. After the loop, SkillEvolver finalizes a deployable skill before $\mathcal{T}_{\text{val}}$ is used for evaluation. Orange steps are run by the SkillEvolver Agent following the meta-skill; blue boxes (Explore, Auditor) are separate CLI-agent sessions.

thoring adds a surface: an agent that reads training labels can encode them into the evaluated artifact. Appendix C describes our controls.

SkillEvolver differs from the work above in combining three choices for the single-task regime: trials diversified by strategies the agent writes, scored by the training variant's verifier, and, when failures remain, a further round in which the skill is refined from how fresh agents fail while using it, with no instance pool and no weight updates. We do not port the methods above as baselines, since most rely on the population this regime lacks; Appendix L discusses what each would reduce to.

3 METHOD

Most methods for improving an LLM agent have it learn from trial-and-error (Shinn et al., 2023). At each iteration r, the agent attempts the training task K times, producing trajectories $\{\tau_{r,i}\}_{i=1}^{K}$ with task rewards $y_{r,i}$ (Song et al., 2024). These trajectories are then analyzed to extract lessons, which methods like Trace2Skill distill into a skill artifact, namely a domain-specific skill, that the agent loads without parameter updates (Ni et al., 2026). SkillEvolver follows this overall pattern with the domain skill itself as the update target. Figure 1 shows the deployment surface, Figure 2 unpacks one iteration, and we provide pseudocode in Appendix A.

3.1 SKILLEVOLVER AS A META-SKILL

SkillEvolver is a *meta-skill*: Rather than solving a target task directly, it instructs a CLI-agent, i.e., the *SkillEvolver Agent*, to author, refine, and deploy a reusable domain skill for that task. We adopt the *meta-X* naming convention following Hu et al. (2025), who introduce a *meta agent* as a foundation model that programs new agents in code. Our setting differs in what plays the meta role. We hold the agent fixed: it can be any CLI-agent that can load skills, such as Claude Code,

OpenHands or Codex. The meta role is instead taken by the *skill*, which instructs the agent to author another skill rather than modify itself.

The learning signal is not the SkillEvolver Agent's reflection on its own execution trajectories, but what a *separate* CLI-agent (given only the candidate domain skill and the task) actually does when handed that skill; we call this second agent the *Domain-Skill Agent*. In each refinement iteration, the SkillEvolver Agent deploys the current candidate skill, spawns multiple Domain-Skill Agents to attempt the task with it, and refines the skill from what the Domain-Skill Agents did or failed to do.

3.2 THE SKILLEVOLVER LOOP

An iteration of the SkillEvolver loop takes a candidate skill v_r and a training task $\mathcal{T}_{\text{train}}$, and returns a refined candidate v_{r+1}, where each candidate is a domain skill, that is, a directory of prose and scripts loadable by any CLI-agent. The refined candidate is obtained by applying a patch to v_r, where the patch is derived from a contrast over labelled trajectories produced by a Domain-Skill Agent running v_r on $\mathcal{T}_{\text{train}}$.

3.2.1 STRATEGY-DIVERSIFIED EXPLORATION

We call our sampling procedure *strategy-diversified exploration*. Before running K parallel rollouts at iteration r, the SkillEvolver Agent writes a strategy set $\mathcal{S}_r = \{s_{r,i}\}_{i=1}^{K}$, where each $s_{r,i}$ specifies a distinct high-level solution plan, and assigns one fresh Domain-Skill Agent to each strategy. Unlike diversity from token-level sampling, this treats diversity as coverage over high-level decision axes such as library choice, algorithm family, and instruction interpretation. The resulting rollout is

$$(\tau_{r,i}, y_{r,i}) = \text{Trial}(v_r, \mathcal{T}_{\text{train}}, s_{r,i}), \qquad i = 1, \ldots, K, \tag{1}$$

where $\tau_{r,i}$ is the trajectory, $y_{r,i}$ is the reward, and $s_{r,i}$ is the strategy assigned to the i-th rollout at iteration r. We use $K{=}4$ throughout the work.

The strategies are not sampled by raising model temperature. Temperature changes local wording and tool-call details, but the resulting agents often share the same high-level plan. Instead, the SkillEvolver Agent writes explicit strategy files before launch and checks that no two strategies are identical on all major axes. A second check marks each concrete training constant as either invariant or parametric; for every parametric axis, at least one strategy must derive the value at runtime rather than copy the training value.

At the first iteration ($r{=}0$) no domain skill has been learned yet, so we deploy a minimal skill that only assigns rollout i to its strategy file $s_{0,i}$: even the bootstrap rollouts are controlled by explicit strategy files rather than left as free-form attempts. At later iterations ($r{>}0$), v_r is already a domain skill, so we place the strategy assignment before the rest of the skill text and update the strategy files to target failure modes observed in the previous iteration.

3.2.2 CONTRASTIVE SKILL UPDATE

Instead of using labeled trajectories to update the agent policy, we use them to update the deployable skill artifact. Given trajectories $\{(\tau_{r,i}, y_{r,i})\}_{i=1}^{K}$ with task rewards $y_{r,i}$, the SkillEvolver Agent compares high-reward and low-reward trials to identify what the current skill is missing. For binary-reward tasks these sets are the passing and failing trials; under a scalar reward they are the top- and bottom-scoring ones, $\tau_r^+ = \text{Top}(\{\tau_{r,i}\}_{i=1}^{K};\, y_{r,i})$ and $\tau_r^- = \text{Bottom}(\{\tau_{r,i}\}_{i=1}^{K};\, y_{r,i})$. The contrastive signal is then summarized as

$$\Delta_r = \phi(\tau_r^+) \setminus \phi(\tau_r^-), \tag{2}$$

where ϕ denotes an LLM-based reading function that extracts task-relevant features from a set of trajectories rather than a programmatic parser. At $r{=}0$, the contrast asks what winners knew that losers lacked. At $r{>}0$, where v_r is already deployed, it instead asks where the skill misled, underspecified, or failed to guide the Domain-Skill Agent. With no success in a batch, Δ_r reduces to a diagnosis of the shared failure modes.

Where Exploration-Based Trajectory Optimization (ETO) (Song et al., 2024) uses such pairs as preference data to update an LLM agent's policy via DPO (Rafailov et al., 2023), we use the contrast as *verbal reinforcement* (Shinn et al., 2023) to refine a frozen-weight agent's skill document, an

instance of artifact-level adaptation (Ni et al., 2026). In compact form, the synthesis step is $\tilde{v}_{r+1} = \mathrm{Patch}(v_r, \Delta_r)$.

The patch is natural-language and code content written into the skill artifact rather than into model parameters; no weights are touched at any iteration, and the main filter is whether a candidate feature would likely be known from pretraining alone, in which case it is not added. Synthesis applies Δ_r as a localized edit. At $r{=}0$, the edit creates the first domain skill v_1 from the contrastive signal and any reusable code observed in high-reward traces. At $r{>}0$, it patches v_r rather than rewriting it, preserving working guidance while adding only the missing constraint, code pattern, or tool exposed by the latest contrast. When executable scripts are included, they must operate on inputs supplied at runtime rather than on filenames, constants, or answers copied from the training instance.

3.2.3 Independent audit and finalization

After synthesis, SkillEvolver invokes an Auditor: a separate CLI-agent session with a clean context, used to verify the candidate skill before it can be accepted into the loop. The Auditor receives only the candidate skill, the task instruction, the training data, and the labelled traces $\{(\tau_{r,i}, y_{r,i})\}$, not validation data or the SkillEvolver Agent's context. As an artifact-level verifier, it checks whether the candidate is self-contained, grounded in observed traces, abstracted away from training-instance constants, and structured so that a fresh Domain-Skill Agent can apply it without relying on the evolver's private reasoning.

The audit returns a binary gate and a set of named violations, $(a_r, E_r) = \mathrm{Audit}(\tilde{v}_{r+1}, \mathcal{T}_{\text{train}}, \{(\tau_{r,i}, y_{r,i})\}_{i=1}^{K})$. The checks are listed in Table 4. They cover both standard overfitting risks (instance-borrowed framing, hardcoded literals, and untraceable claims) and deployment-specific risks, such as whether strategy axes are abstracted, whether a primary script is surfaced where a Domain-Skill Agent will read it, and whether the skill can be silently bypassed.

A clean audit promotes the candidate to the next accepted skill, $v_{r+1} = \tilde{v}_{r+1}$, whereas a critical finding triggers a targeted re-patch, at most twice (Appendix K). Whether to iterate again is the agent's own decision, guided by a reference rule rather than a hard threshold: stop once the accepted skill audits clean and at least $3K/4$ of the latest trials pass, or when the budget is exhausted. Before validation, SkillEvolver selects from the accepted skills $\{v_j\}_{j=1}^{r+1}$, optionally merges them itself, and uses training pass rate, trace cost, and generalization risk to choose the artifact written to the deployment directory for validation on $\mathcal{T}_{\text{val}}$.

4 Experiments

4.1 Setup

We evaluate on SkillsBench (Li et al., 2026b), 87 tasks across 8 professional domains scored by the benchmark's own verifiers.[1] We run DeepSeek-V4-Flash and GLM-5.2 under OpenHands, the harness behind the official leaderboard, and Claude Opus 4.6 under Claude Code; because of its cost, we evaluate Claude Opus 4.6 on a random 60-task subset. The SkillEvolver Agent and the Auditor run under Claude Code by default and always use the same backbone as the Domain-Skill Agent they author for, so each block of Table 1 involves a single model end to end.

Each model is evaluated under three conditions on the same harness and host: **A**, no skill; **B**, the human-curated skill shipped with the task; **C**, the skill SkillEvolver distils on the task's *training* variant, which never sees the validation instance. For all three models we run A and B ourselves rather than quoting leaderboard rows, so every comparison is paired and within-configuration, at 87 tasks (60 for Claude Opus 4.6) $\times$ 3 validation trials per arm.

A rollout that hit the harness wall-clock is an infrastructure outcome rather than a model failure and is excluded; a task an arm left unmeasured is never folded in as a zero, and no cell is imputed. Because task difficulty dominates across-task variance, arms are compared on the paired per-task difference. DeepSeek-V4-Flash was swept three times end to end, giving 55.8 ± 3.05 across sweeps. Definitions, per-arm coverage and the authoring budget are in Appendix F.

[1] Release `v1.1`, commit `b63b7b28`.

Table 1: Resolution rate (%) on SkillsBench v1.1, by model and harness (left) and domain, with task counts under each header: software engineering, industrial and physical systems, office and white-collar work, natural science, finance and economics, mathematics and formal reasoning, cybersecurity, media content production. **A**: no skill. **B**: the task's human-curated skill. **C**: the skill SkillEvolver distils on its training variant. *Official* rows quote the published leaderboard entry for the same model and harness, an aggregate over all 87 tasks (none for GLM-5.2); Opus 4.6 uses the random 60-task subset of §4.1; nothing is imputed (Appendix F). *Gain* is Δ, a condition's overall rate minus that of A, with the leaderboard's headroom-normalised gain $g = \Delta/(100 - \mathrm{A})$ in parentheses. $C - B$ is the difference between the overall rates of C and B, with a 95% interval from the per-task paired difference; bold: best per column per model.

		SWE (16)	IPS (14)	OWC (14)	NSci (14)	Fin (9)	Math (8)	Cyb (7)	Med (5)	Overall	Gain (Δ (g))	$C - B$ (95% CI)
DeepSeek OpenHands	Official A	–	–	–	–	–	–	–	–	27.5		**+13.3** [+1.7, +24.9]
	Official B	–	–	–	–	–	–	–	–	44.7	+17.2 (23.7)	
	A (none)	30.4	28.6	39.3	32.5	11.1	31.2	0.0	10.0	26.3		
	B (curated)	**46.0**	35.7	42.9	67.3	22.2	**62.5**	28.6	40.0	44.6	+18.2 (24.8)	
	C (ours)	39.3	**66.7**	**52.4**	**70.7**	**51.9**	54.2	**59.5**	**86.7**	**57.9**	**+31.5 (42.8)**	
GLM OpenHands	Official	–	–	–	–	–	–	–	–	–	–	+7.2 [−4.1, +18.4]
	A (none)	42.9	21.4	50.0	50.6	11.1	25.0	0.0	20.0	31.7		
	B (curated)	62.2	**60.7**	**64.3**	64.3	22.2	50.0	35.7	60.0	55.0	+23.4 (34.2)	
	C (ours)	**70.9**	53.6	54.8	**66.7**	**59.3**	**62.5**	**65.9**	**66.7**	**62.2**	**+30.5 (44.7)**	
Opus 4.6 Claude Code	Official A	–	–	–	–	–	–	–	–	33.7		**+10.8** [+1.2, +20.4]
	Official B	–	–	–	–	–	–	–	–	50.2	+16.5 (24.9)	
	A (none)	31.5	37.0	37.5	62.4	22.2	44.4	28.6	33.3	38.7		
	B (curated)	**51.9**	55.6	58.3	57.1	38.9	44.4	60.2	66.7	54.7	+16.0 (26.2)	
	C (ours)	46.7	**60.0**	**73.8**	**81.4**	**40.0**	**63.3**	**68.6**	**93.3**	**65.5**	**+26.8 (43.8)**	

4.2 Main Results

Results are presented in Table 1. On all three models, the skills learned by SkillEvolver outperform both the no-skill baseline and the human-curated skills, improving over the curated skills by 13.3, 7.2 and 10.8 points on DeepSeek-V4-Flash, GLM-5.2 and Claude Opus 4.6, respectively. Note that our curated baselines are not weakened re-implementations: on DeepSeek-V4-Flash they reproduce the published leaderboard to within 1.2 points, and on Claude Opus 4.6 they sit above it.

The improvement is significant on DeepSeek-V4-Flash and Claude Opus 4.6, and not on GLM-5.2, whose curated skills are already the strongest of the three. On Claude Opus 4.6, A and B sit a few points above the leaderboard's aggregates, which are not directly comparable (60 vs. 87 tasks, and a newer Claude Code release).

Per-domain gains over the curated skills differ by model. On DeepSeek-V4-Flash, the largest is in media content production (+46.7), followed by industrial and physical systems (+31.0), cybersecurity (+30.9) and finance (+29.7); curated skills stay ahead in software engineering and mathematics. On GLM-5.2, the largest is in finance (+37.1), where curated skills add only 11.1 points over no skill, followed by cybersecurity (+30.2); curated skills stay ahead in industrial and physical systems and in office and white-collar work. On Claude Opus 4.6, the largest are in media content production (+26.6) and natural science (+24.3), and software engineering is the only domain where curated skills stay ahead. Software engineering is thus the one domain where curated skills lead on two of the three models.

4.3 What the Loop Contributes

Components. To evaluate the importance of each component, we remove one at a time and re-run the loop from scratch on DeepSeek-V4-Flash, with the same authoring budget as in Table 1. Each removal is compared with the full method on its own random subset of tasks, so levels are not comparable across rows. Results are presented in Table 2 (top).

Table 2: What the loop contributes. *Top*: component ablation on DeepSeek-V4-Flash; each row is a paired difference on its own randomised subset of n tasks, so levels are not comparable across rows. *Bottom*: how the last 4 of 8 rollouts are spent, on the 51 tasks whose first 4 all failed; both plans spend the same 8. *rollouts*: the share of those four that pass; *batches*: the share of four-rollout batches with at least one success. The last row keeps the 30 tasks whose training oracle was unread before the second batch.

Removed	**full**	**ablated**	Δ	n
Strategy injection	56.6	45.7	$\mathbf{-11.0}_{[-20.4,\,-1.8]}$	73
Auditor	52.8	50.9	$-1.9_{[-10.7,\,+7.0]}$	36
Failure traces	52.8	47.2	$-5.6_{[-14.1,\,+3.0]}$	36
Last 4 of 8		**rollouts**	**batches**	
Blind (K=8)		8.0%	29.7%	
After distillation (2×4)		**50.0%**	**64.4%**	
paired Δ		$+42.0_{[+30.6,\,+53.5]}$	$+34.6_{[+19.2,\,+50.1]}$	
oracle withheld (n=30)		14.5 → 59.0 $_{+44.5[+15.5,\,+73.5]}$	23.3 → 60.0	

We first examine strategy injection. Without written strategies, the K exploration rollouts become independent samples of one prompt, and the score drops by 11.0 points, about as much as the entire gain of the distilled skill over the curated one in Table 1. The traces show why. Temperature changes the surface form of a model's code, not its choice of method: before we introduced strategies, all four trials on `earthquake-plate-calculation` chose the same map projection and returned the same wrong distance, leaving nothing to contrast. A written strategy moves the variation to the approach itself and labels it, so that a failure can be traced to a stated design decision. It is also the component that hands the most control to the agent, which writes the strategies, reads their outcomes and decides what to try next. That the largest effect comes from this component, rather than from one that encodes our prior about good skills, is in the spirit of the bitter lesson (Sutton, 2019).

Next, we withhold the failing trajectories from the distiller. The score drops by 5.6 points, though not significantly at this subset size. Where failures are all there is, their value is direct. Skills distilled from batches in which every trial failed still work: a second batch then solves the training task 64.4% of the time, against 29.7% when drawn blind on the same tasks (Table 2). The distiller also cites failing traces about twice as often as passing ones. We therefore treat failures as an indispensable input.

Finally, removing the auditor changes accuracy by only 1.9 points, and we do not count it as a contribution. We keep it for a failure mode that a pass rate cannot see. Early distilled skills often carried defects obvious to a human reader: an answer from the training instance hard-coded as a literal, a script that reads its input by fixed row and column, a skill named after the training cover story (Appendix K). SkillsBench scores a skill on one held-out instance, so a brittle skill that happens to fit it scores the same as a robust one; on inputs further from the training instance, such defects fail silently. The auditor removes them: 39 of 42 flagged training-only literals are absent from the delivered skills (Appendix I).

The second iteration. Does refining a skill help more than drawing more rollouts? Comparing R=2 with R=1 cannot tell, since R=2 also draws K more rollouts. We therefore fix the budget at eight rollouts and vary only how they are spent: all eight blind, or four, then distillation, then four more. The blind plan comes from our three repeated sweeps, whose opening batches are written before the loop has read any trajectory, and the two plans are compared on the 51 tasks where the first four rollouts all failed. Results are presented in Table 2 (bottom). Spent blind, four more rollouts rescue such a task about once in three; spent after distillation, about twice in three. The result holds when the training oracle is withheld until the second batch. The full method is also the cheaper plan, since the agent stops after one iteration when its first batch is already good: it spends 6.57 exploration rollouts per task on average, against a flat eight. These are pass rates on the training variant rather than held-out accuracy. K is chosen in §4.4; the cap R=2 balances cost against Appendix H.

Table 3: The two curves behind $K{=}4$, enumerated on the $K{=}8$ run. Marginal pass rate per 1M tokens has flattened by $K{=}4$; the contrast a distiller reads is still climbing.

K	1	2	3	**4**	5	6	7	8
P(success)	.509	.584	.619	**.642**	.661	.679	.696	.714
P(contrast)	.000	.151	.226	**.279**	.320	.357	.393	.429
Δpass / 1M tok	.579	.086	.039	**.026**	.022	.020	.020	.020

4.4 Choosing the Batch Size

The batch size K is the one setting whose cost grows linearly, so we choose it from data rather than tune it. A batch is most useful to distillation when it contains a success and, since the skill is written from the difference between trajectories, a success and a failure together. We compute the probability of each exactly from the exploration rollouts we collected, using a dedicated $K{=}8$ run on DeepSeek-V4-Flash (14 tasks), the only data past $K{=}4$ (Appendix H). Results are presented in Table 3.

The two curves behave differently. The chance of a success saturates early: beyond $K{=}4$, each additional rollout adds less than two points at the same token cost, so a larger batch is never a better deal. The chance of a contrast, which is what distillation consumes, is still rising steeply at $K{=}4$, and at $K{=}2$ fewer than one batch in six contains any contrast at all. We therefore take $K{=}4$, where the cheap coverage is used up but the contrastive signal has not yet flattened.

5 Conclusion

We presented SkillEvolver, a meta-skill that lets a CLI-agent author, deploy and refine a skill for a single new task from a handful of trials, learning the skill's prose and code rather than model weights. Two design choices make so small a budget enough: each trial follows a distinct strategy the agent writes, and, when a first batch leaves failures, the skill is refined from how fresh agents fail using it.

On SkillsBench, the skills learned by SkillEvolver outperform the human-curated skills in overall resolution rate on all three models we evaluate, and the improvement is significant on DeepSeek-V4-Flash and Claude Opus 4.6. Strategy injection accounts for the largest share of the gain, and four rollouts spent after distillation rescue a dead-ended task about twice as often as four spent blind.

We plan to apply SkillEvolver to tasks with continuous rewards, where a trial carries more signal than a discrete score, and to organize the per-task skills it produces into a library that can be shared, deduplicated and maintained across tasks.

Limitations

Model and harness coverage. The headline numbers come from three models, two open-weight models under OpenHands and Claude Opus 4.6 under Claude Code; three backbones are too few to claim the method is insensitive to the backbone, and how often a model reads a mounted skill at all varies sharply across models (Appendix G).

Refinement depth not characterized. The cap $R{=}2$ balances cost against Appendix H; it is not a measured optimum. Our evidence that the second iteration does targeted work is the compute-matched reconstruction of §4.3, which measures exploration pass rate on the training variant rather than held-out accuracy, and we did not run $R{\geq}3$ at scale.

Component effects at ablation scale. Removing the auditor or the failure traces lowers the score, by 1.9 and 5.6 points, but neither interval excludes zero at our subset sizes (§4.3), so we do not list the auditor as a contribution.

Benchmark and task scope. We evaluate only on SkillsBench, whose rewards are discrete rather than continuous, so we have no evidence for continuous or shaped rewards; nor do we address how a population of per-task skills is organized, deduplicated or maintained.

AI USE STATEMENT

Large language models are the object of study in this paper: the SkillEvolver Agent, the Domain-Skill Agents, the Auditor and every evaluated backbone are LLM agents, as described in §3 and §4. Beyond that role, we used LLM coding assistants to implement and debug parts of the experiment infrastructure and analysis scripts; every reported number is produced by those scripts from the raw run logs, and the authors checked them against the logs. We also used LLM assistants to edit the manuscript for clarity, to format the LaTeX source, and to check each bibliography entry against its primary source. The training variant of each task (Appendix D) was generated with LLM assistance, and every variant was then checked and validated by the authors. LLMs were not used to formulate or prove mathematical claims, or to propose the research questions. The authors reviewed all AI-assisted content and take full responsibility for it.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems. arXiv preprint arXiv:2603.02766, 2026.

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

Anthropic. Claude code: An agentic coding tool. https://github.com/anthropics/claude-code, 2025b.

Anthropic. skill-creator. https://github.com/anthropics/skills/tree/main/skills/skill-creator, 2025c. Agent Skill in the GitHub repository anthropics/skills. Accessed: 2026-03-30.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. arXiv:2408.08435.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. XSkill: Continual learning from experience and skills in multimodal agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

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 (ICLR)*, 2024.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. *arXiv preprint arXiv:2603.02176*, 2026a.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-chung Lee. SkillsBench: Benchmarking how well agent skills work across diverse tasks. arXiv preprint arXiv:2602.12670, February 2026b.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. SkillGen: Verified inference-time agent skill synthesis. arXiv preprint arXiv:2605.10999, 2026a.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. SkillClaw: Let skills evolve collectively with agentic evolver. arXiv preprint arXiv:2604.08377, 2026b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

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.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. 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 *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.25140.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023. doi: 10.52202/075280-2338.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.409.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR, 2020. URL https://proceedings.mlr.press/v119/sun20b.html.

Richard S. Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

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

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

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. *arXiv preprint arXiv:2602.08234*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

A SkillEvolver Full Pseudocode

Figure 2 in the main text gives the high-level pipeline; Algorithm 1 below makes the strategy-diversified sampling and deploy-then-refine loop explicit.

B Auditor check list

The Auditor subagent (§3.2.3) runs the ten mechanical checks in Table 4. Checks 1–6 cover standard content-level leakage patterns; Checks 7–9 target failure modes specific to deploying a skill to a separate agent (parametric-axis under-abstraction, primary-action hoisting, silent-bypass). Check 9 is evaluated on the Domain-Skill Agents' traces; Checks 7 and 8 are static checks on the skill text, for failure modes we first observed in those traces.

Algorithm 1: SkillEvolver. The key design choice is *strategy-diversified sampling on a deployed skill*: every iteration writes K strong-prior strategies (line 2 at $r{=}0$, line 9 at $r{>}0$) that the harness routes to parallel trials via a per-trial environment index (lines 3 and 10); from $r{=}1$ onwards those strategies are aimed at the current candidate skill v_r's observed weak spots rather than at the original decision axes. Line 7 is the SkillEvolver Agent's own decision whether to run another iteration; the stated rule is a reference the meta-skill gives the agent, not a hard threshold. Line 8 is the second half of the commitment: v_r is installed as a real dependency in the trial container, so the contrast at line 11 reflects where a fresh using-agent was actually helped or misled by the deployed skill. `AuditRepair` (lines 5 and 12) runs the fresh-session auditor of §3.2.3 on the new candidate; a clean audit accepts it, and a critical finding triggers a targeted re-patch against the reported violations, at most twice, after which the candidate continues with its audit record (Appendix K). Line 12 patches rather than rebuilds from scratch, and line 14 selects among, or merges, the skills produced so far by training pass rate, trace cost and generalization risk.

Input : task $\mathcal{T}$; iteration cap R; explore width K; validate trials V
Output : $\pi(v^*; \mathcal{T}_{\text{val}})$

1 axes $\leftarrow$ `Parse(`$\mathcal{T}_{train}$`)` `// §3.2.1 one-shot decision axes`
2 $\mathcal{S}_0 \leftarrow$ `DiverseStrategies(`*axes,*$\varnothing$*,*$\varnothing$`)` `// §3.2.1 bootstrap: K strong-prior strategies routed by trial index`
3 $\tau_0 \leftarrow$ `Explore(`$\mathcal{T}_{train}, \mathcal{S}_0, v{=}\varnothing, K$`)` `// §3.2.1 K parallel trials; trial i loads` $s_{0,i}$ `via env index`
4 $\Delta_0 \leftarrow$ `Contrast(`τ_0^+, τ_0^-`)` `// §3.2.2 winners \ losers; failure diagnosis if` $\tau_0^+{=}\varnothing$
5 $v_1 \leftarrow$ `AuditRepair(Distill(`Δ_0`))`; **mirror** v_1 to `output/` `// §3.2.3 failsafe deploy copy`
6 **for** $r \leftarrow 1$ **to** $R{-}1$ **do** `// §3.2.1 deploy-then-stress-test loop`
7 **if** *agent judges* τ_{r-1} *sufficient (reference:* v_r *audit-clean* $\wedge$ *#pass*$(\tau_{r-1}) \geq 3K/4$*)* **then break** `// §3.2.3 agent's continue-or-exit decision`
8 **deploy** v_r as a live skill in the trial container `// §3.2.1` **`real-scenario sampling`**
9 $\mathcal{S}_r \leftarrow$ `DiverseStrategies(`*axes,*v_r, τ_{r-1}`)` `// §3.2.1` **`persists:`** `K stress-test priors aimed at` v_r`'s weak spots`
10 $\tau_r \leftarrow$ `Explore(`$\mathcal{T}_{train}, \mathcal{S}_r, v_r, K$`)` `// §3.2.1 trial i loads` $s_{r,i}$ `first, then consults` v_r
11 $\Delta_r \leftarrow$ `Contrast(`τ_r^+, τ_r^-`)` `// §3.2.2 where did` v_r `mislead?`
12 $v_{r+1} \leftarrow$ `AuditRepair(SurgicalPatch(`v_r, Δ_r`))` `// §3.2.2, §3.2.3 surgical patch, then audit`
13 **end**
14 $v^* \leftarrow$ `Finalize(`$\{v_j\}_{j=1}^{r+1}$`)` `// §3.2.3 select or merge (no harness call)`
15 **return** `Validate(`$v^*, \mathcal{T}_{val}, V$`)` `// §3.2.3 held-out validation`

C CONTAMINATION CONTROLS

Layer 1: train/test split. All iterations of the evolve loop run on $\mathcal{T}_{\text{train}}$, a generated variant with different data, filenames, and sometimes sub-domain from $\mathcal{T}_{\text{val}}$; validation runs on $\mathcal{T}_{\text{val}}$. A skill that encodes a training-specific filename or value silently fails validation – the file is not there. Before each exploration run, the curated training skill is deleted at source, so it is never reachable. Contamination discipline in this form follows precedent in LLM evaluation (Sainz et al., 2023); the agent setting adds a new surface, since the authoring agent may itself encode training-label content into the skill artifact.

Layer 2: workspace whitelist. A PreToolUse hook denies every agent tool call outside a single per-run workspace prefix. The validation task directory, curated validation skill, and test suites live outside and are unreachable. A denylist tripwire checked before the whitelist denies `..` traversal and any path resolving into the curated training-skill slot. Path resolution checks both the raw and symlink-resolved paths.

Table 4: Auditor checks. ⋆ marks critical checks, whose findings can block acceptance and force a targeted re-patch (Algorithm 1). Checks 1–6 target content-level overfit (standard leakage patterns). Checks 7–9 target failures specific to a deployed skill: parametric-axis under-abstraction, structural failure to hoist the primary action, and silent-bypass at runtime. Check 9 reads the traces of the candidate skill as a live dependency (§3.2.1); Checks 7 and 8 inspect the skill text for failure modes first observed in such traces.

#	Check	What it catches
1	Framing⋆	`name`/`description` borrows a training-instance business noun instead of the abstract operation.
2	Literals⋆	Hardcoded training filenames, field names, entity strings, or soft-qualifier numericals ("`typically` $<$ 2.5").
2b	Script bloat	Individual script exceeds 200 lines (important) / 400 lines (critical), almost always a bundled solution generator rather than a focused utility.
3	Untraceable	Imperative assertions ("use X not Y", "never", "required") without trace provenance and not pretraining-obvious.
4	Shape-bake⋆	Scripts index by hardcoded column, sheet, or key without a runtime probe (`df.columns`, `wb.sheetnames`); ≥ 3 `if "keyword" in desc` branches in one script flag keyword-dispatch solution-generators.
5	Coverage	Mechanical task with zero bundled scripts (skipped in high-pass mode).
6	X-ref⋆	Any $\geq$ 4-char string literal in skill files matching a training filename, field, or value.
7	Under-abstraction⋆	Imperative claims embedding a parametric-axis constant (e.g. terminal-state code, threshold) without a sibling "re-derive at runtime" instruction or invariance provenance.
8	Primary-action hoisting⋆	Skill declares a `primary_script` but `SKILL.md` routes constraints/background prose *before* the invocation block, so the using-agent reads constraints first and never invokes the script.
9	Silent-bypass⋆	Skill declares a `primary_script` but exploration traces contain zero Bash invocations of it under majority-fail, so a correct skill is quietly ignored at runtime.

D Training-Variant Generation Methodology

Training variants $\mathcal{T}_{\text{train}}$ are disjoint from the corresponding $\mathcal{T}_{\text{val}}$ on filenames and data values, sometimes on sub-domain. Variants are generated per task by a separate LLM-assisted pipeline that preserves task structure (the same test specification format, the same expected output shape) while resampling inputs, and every variant is then checked and validated by the authors; the curated training skill is deleted at source, so it is never reachable during exploration. The generation pipeline is orthogonal to SkillEvolver (at any R); we use its validated output as-is.

E Per-Task Results

Figure 3 gives every per-task rate behind Table 1, for the two OpenHands models, in the same heatmap form as Figures 11 and 12 of the SkillsBench paper. For SkillEvolver, per-task Finalize choice (v_1 vs v_2 vs merge), per-phase scores (exploration, refinement, validation), and per-task token and dollar cost are recorded for every run in the released results database.

Grey marks a task an arm never measured. Nothing is imputed, either here or in Table 1, so the column means of this figure are exactly the rates reported there; the figure is built by importing the scorer that produces the table, so the two cannot drift apart.

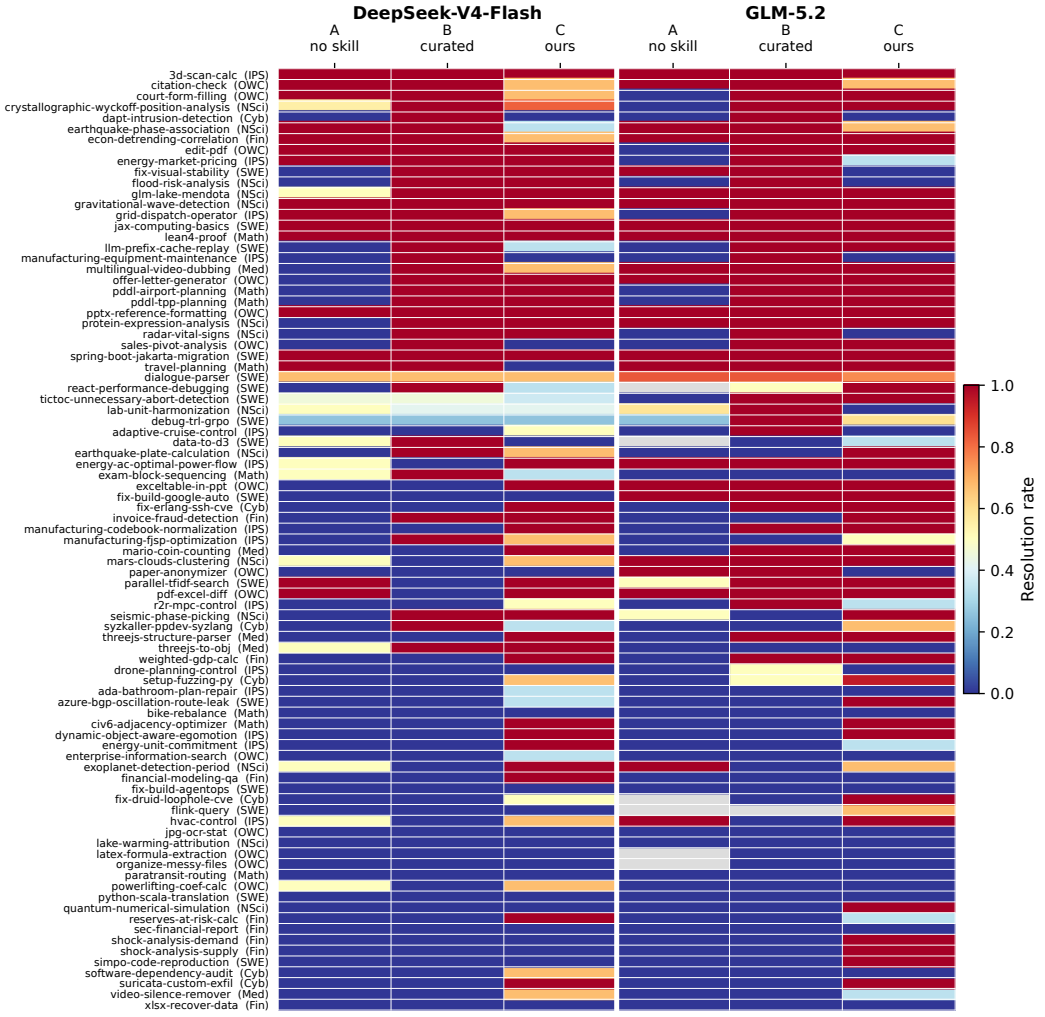

Figure 3: Per-task resolution rate on all 87 SkillsBench v1.1 tasks under OpenHands, for the two OpenHands models of Table 1. **A**: no skill. **B**: the human-curated skill shipped with the task. **C**: the skill SkillEvolver distils on the task's training variant. Rows are sorted by the mean of the two curated columns, descending, so that neither model orders the figure on its own. Grey cells were never measured by that arm and are left un-imputed. Domain tags follow Table 1.

F Evaluation Protocol and Authoring Budget

Metrics. For held-out evaluation (Table 1) we define no metric of our own: the resolution rate and gains are the ones the official SkillsBench leaderboard reports, computed the same way. Tables 2 and 3 additionally report training-variant pass rates, batch-level rescue rates, $P(\text{contrast})$ and pass rate per 1M tokens, each defined where it is used. The per-task *resolution rate* is the mean reward over that task's validation rollouts (up to 3 trials). The aggregate is the unweighted mean of per-task rates. A rollout that hit the harness wall-clock is an infrastructure outcome and is excluded; a task an arm did not measure is never folded in as a zero and is never imputed, so an arm is scored over the tasks it actually covered, and the number of tasks behind each rate is stated wherever it is not 87. Tasks missing from our records are marked grey in Figure 3. Per-condition rates, with 95% normal-approximation half-widths using the *task* as the unit ($1.96\,\mathrm{sd}/\sqrt{n}$, the leaderboard's convention) and the number of tasks each covers, are 26.3 ± 8.3 (87) / 44.6 ± 10.3 (87) / 57.9 ± 8.7 (87) for DeepSeek-V4-Flash, 31.7 ± 9.9 (81) / 55.0 ± 10.4 (86) / 62.2 ± 9.4 (87) for GLM-5.2, in the order A / B / C. Because task difficulty dominates this variance, such intervals overlap by construction, as they do on the published rows as well, which is why the main text compares arms on the paired per-task difference instead. Δ is the difference of two aggregate rates in points and $g = \Delta/(100 - \mathrm{A})$ is the leaderboard's headroom-normalized gain.

Repetitions. DeepSeek-V4-Flash was swept over the full roster three times; GLM-5.2 once. The three DeepSeek sweeps are complete independent repetitions of the whole pipeline (exploration, distillation, refinement and validation), not validation replays of one skill, and give a mean and standard deviation across sweeps of 55.8 ± 3.05. This is not the same quantity as the single-sweep across-task interval reported above.

Authoring budget. SkillEvolver uses $K{=}4$ exploration trials per iteration and at most $R{=}2$ iterations, a nominal cap of 11 rollouts per task. $R{=}2$ is a cap rather than a schedule: the agent stops after the first iteration when that batch is already good, giving a measured 6.57 exploration rollouts per task, a mean effective R of 1.64, and 9.57 rollouts per task including the three validation trials. Against the compute-matched $R{=}1$ at $K{=}8$ alternative the full method therefore spends less: 6.57 against 8.00 exploration rollouts.

G Skill Invocation Rates

Whether a mounted skill is read at all varies sharply by model. In the curated condition, the task's own bundled skill is invoked in 47.1% of DeepSeek-V4-Flash rollouts and 80.5% of GLM-5.2 rollouts. On our distilled skill the corresponding rates are 64.6% and 89.3%: both models read the distilled skill more often than the curated one, but the ordering across models is unchanged. A skill that is never read cannot help, and the effect is confined to the two conditions in which one is mounted.

H Batch Size: Enumeration Bases and the Third Iteration

Enumeration bases. The curves in Table 3 are enumerated, not simulated: for each task we count its healthy exploration rollouts N and its successes S, and evaluate $1 - \binom{N-S}{K}/\binom{N}{K}$ and its two-sided form exactly, then average over tasks. Past $K{=}4$ this requires a task to have eight rollouts, which only the dedicated $K{=}8$ arm has (DeepSeek-V4-Flash, 14 tasks, 112 trials, $N{=}8$ per task), so the table is read off that arm; the production batches collected across both models (676 healthy exploration trials over 169 task-runs with four rollouts, DeepSeek-V4-Flash 86 / GLM-5.2 83) agree with it on shape over the range where both are defined, at $P(\text{success}) = .343/.426/.472/.503$ and $P(\text{contrast}) = .000/.166/.249/.302$ for $K = 1 \ldots 4$. The $K{=}8$ arm sits at higher absolute coverage because its tasks are easier than the roster average, which is why we build no validation claim on it. Cost is 879k–1.24M tokens per trial, and wall-clock barely grows with K, since trials run concurrently.

Why not a third iteration. A third iteration would need both remaining failures to learn from and failures within reach; the first holds for only about half of runs, and the second barely at all. After

k=1, 55.7% of runs still contain a failing trial (45.8% once timeout-induced failures are removed), and multiplying by the 89.2% delivery retention rate bounds the fraction of runs with anything left to distil at roughly 50%. Of the runs that are genuinely stuck (19.2%, still all-failing after k=1), only 1 of their 123 trials ever scored non-zero: these are hard zeros rather than near-misses.

I Memorisation Audit

A training variant re-instantiates the same *method* over different data. On `econ-detrending-correlation`, training uses government spending × exports, 1965–2019, deflated by PPI, from `ERP-2020` tables, while validation uses personal consumption × private fixed investment, 1973–2024, deflated by CPI, from `ERP-2025` tables. Only the HP filter, log-transform, λ=100 and answer format are shared, so every filename, column, year range and even the deflator differs.

Of the critical literals that the auditor flagged and that provably occur only in the training variant (instance identifiers such as `P-SANR-SHWER` or `CM-4471902337`) 92.9% (39 of 42) are absent from the delivered skill, Clopper–Pearson 95% $[0.805, 0.985]$. A coarse hard-coded-answer regex (`answer = <num>`, `return <float>`, `expected_output =`) fires 0 times across all 167 delivered skills.

J Strategy Prescriptions: Delivery and Execution

Measurement uses trace text only, the text after `## Turn 1` or inside `### Tool Call:` blocks, because the prompt portion of a trajectory contains the prescription verbatim and would yield a spurious self-recall of 1.0.

Delivery is exact: $528/528$ trials in injected iterations carry the prescription block, against $0/516$ in un-injected ones. Execution: recall of a trial's own strategy tokens exceeds that of its sibling trials by $+18.5$ and $+17.6$ points on DeepSeek-V4-Flash and GLM-5.2 respectively (all $p < 5 \times 10^{-5}$), and a library or CLI named by exactly one prescription is executed by the assigned trial at rate 0.919 against 0.549 by its siblings on DeepSeek-V4-Flash ($+0.370$).

The four prescriptions are not one prescription sampled four times. Replacing them with a byte-identical single one raises the Jaccard similarity of command tokens within a batch from 0.139 to 0.274 (paired $+0.135$ $[+0.078, +0.207]$, $p = 0.0005$, $n = 13$), an effect that grows when recomputed on only the first 5, 10 or 20 commands of each trial, against a same-window drift bound of $+0.017$–$+0.086$.

One negative control belongs with this evidence. Among mixed batches, the rate of executing a practice named only in one's own prescription is nearly identical for losing trials (55.0%, $22/40$) and winning ones (59.2%, $29/49$). Prescriptions change behaviour, but changed behaviour does not by itself explain winning; these cases are mechanism illustrations rather than effect sizes.

K Auditor Gating Rates

Over 203 audit records, a check "fires" when it produces at least one finding and "gates" when that finding is critical, blocking acceptance and forcing a repair (Table 5). 16.3% of audits contain at least one critical finding. The distribution is informative about the architecture: the four leakage-family checks (hardcoded-literal, cross-reference, shape-bake, framing) account for 72.3% (73 of 101) of all critical findings, while each remaining quality check gates at $\leq 2.0\%$ even when it fires in up to a third of audits. The auditor is therefore better described as four gates plus a set of advisory findings passed to the next synthesis step than as ten equal checks. Severity distribution: important 295, informational 248, critical 101, minor 5.

Leaks it blocked. On `earthquake-plate-calculation` the skill's output example hard-coded the training instance's answer itself (`boundary_distance_km: 2086.56`, with matching magnitude and coordinates) rather than deriving it; such a skill hands the downstream agent a number instead of a procedure. On `protein-expression-analysis` the skill named *itself* after the training cover story (`metabolite-abundance-analysis`, with a matching description and script name) while the validation task is protein expression: what leaks is the framing, and

Table 5: Per-check firing and gating rates over 203 audit records.

Check	fires	of which critical
untraceable-claim	33.0%	1.0%
hardcoded-literal	33.0%	**7.4%**
cross-reference	27.6%	**6.9%**
under-abstraction	27.1%	1.5%
silent-bypass	20.2%	2.0%
script-bloat	18.7%	1.5%
script-coverage	9.4%	1.0%
shape-bake	4.9%	**2.0%**
primary-action-hoisting	3.0%	0.0%
framing	2.5%	**2.0%**

the cost is that the downstream agent may never recognise the skill as applicable. After the name and description were required to describe the abstract operation, the skill scored $3/3$ on validation. On `sales-pivot-analysis` the bundled script addressed its input by *position* rather than name (`row_idx=2`, `data_idx=3`) with no runtime probing, so on a validation instance whose columns sit elsewhere it does not crash; it silently reports wrong numbers. The repair replaced fixed indices with runtime column discovery.

Two concessions. The auditor's incremental contribution to final accuracy is not measurable at the sample size we could afford: the ablation in Table 2 is the only estimate we have and it is a null, so we do not claim the auditor as a contribution (§4.3), and report it here for what it demonstrably does, which is to remove leakage patterns from the delivered artifact. The design also deliberately ships imperfect skills: after two failed repair attempts within an iteration the run continues with a machine-readable audit record, and when this happens on the final iteration the skill is delivered with unresolved violations and an explicit record of what remains unfixed.

L Positioning Against Concurrent Skill-Learning Methods

Table 6 expands the comparison summarized in §2.

Unit of evidence. Every method in the table learns from a population of instances, queries or users, and a single task does not supply one. Ported to it, the cross-task parts lose their data: XSkill's library and ReasoningBank's bank would hold only what one training variant yields, and the gates of EvoSkill and SkillGen would decide from a pass rate over three trials, which moves in steps of 33 points. The within-task parts survive: XSkill's cross-rollout critique and ReasoningBank's parallel mode still contrast temperature-sampled trials of one instance, which is our pipeline without strategy injection, the ablation that costs 11.0 points (§4.3). We did not run these ports as baselines.

Where the loop closes. EvoSkill, SkillGen, SkillClaw and skill-creator, like SkillEvolver, refine a skill from the behaviour of an agent that holds it, so this alone does not distinguish our method. What differs is the evidence the refinement reads and who decides the next step. The other methods aggregate that behaviour over many instances or users, or rely on a human to judge it; SkillEvolver reads the failures of a handful of trials on one training variant, and the agent decides whether to explore again, what to prescribe next, or to stop. This is why runs average 6.57 exploration rollouts against the 8 of a fixed plan (§4.3). The loop is bounded by the rules the meta-skill writes down, so it is guided rather than open-ended search.

Table 6: How trajectory-derived skill learning is set up. The second column is what a method must be given before it can start; SkillsBench presents 87 independent tasks, with no sibling instances, no shared stream and no user fleet. Scales are as reported by each paper.

Method	**Evidence it requires**	**Trial diversity**	**What closes the loop**
Trace2Skill (Ni et al., 2026)	a labelled pool per domain (e.g. 400 spreadsheet problems split 200/200, one trace each)	not a rollout sampler	a single-pass merge; the skill is not re-deployed
XSkill (Jiang et al., 2026)	100 sibling training tasks $\times$ 4 rollouts	temperature 0.6	one accumulation pass in the reported experiments
ReasoningBank (Ouyang et al., 2026)	a stream of related queries, self-judged, no labels	temperature 0.7; parallel mode samples one query k times	the next query in the same stream
EvoSkill (Alzubi et al., 2026)	a labelled dataset split into train, validation and test (e.g. 12–36 training and 17 validation questions on OfficeQA)	none at rollout level	executor failures propose edits; a validation score admits them
SkillGen (Ma et al., 2026a)	a construction set split 70/30 into induction and verification	temperature 0	up to eight refinement rounds, each verified with and without the skill on a held-out 30%; the best candidate is kept
SkillClaw (Ma et al., 2026b)	8 simulated users $\times$ 6 days of sessions	sessions across users	an agentic evolver; re-execution against the incumbent
SkillEvolver (ours)	one task, one generated training variant with its verifier, 6.57 exploration rollouts	strategies written by the agent	a fresh agent's failures while using the skill, plus an audit