SELFEVOSKILL: PAIRED EXECUTION AUDITS FOR SKILL REVISION WITHOUT EXTERNAL OUTCOME SUPERVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills provide LLM agents with reusable procedures, but a given skill may be incomplete, ineffective, or misleading for the current task. Existing methods for revising skills typically rely on external outcome supervision, such as task rewards, performance on a separate validation set, or feedback from deployment, to decide which updates to retain. However, this supervision may be costly or unavailable when adapting a skill to a new task. Using execution behavior as an alternative signal is challenging: a single run reflects the combined effects of the agent, task, and workspace, making the skill's contribution difficult to isolate. Comparing executions with and without the skill, while holding these factors fixed, provides evidence about how the skill changes behavior. We introduce **SELFEVOSKILL**, which links these behavioral differences to relevant skill passages and iteratively revises and reevaluates those passages through subsequent executions. Checks derived once from the task's explicit requirements prevent revisions from violating task constraints. Across 89 containerized tasks in 8 professional domains, under the same best-of-two evaluation budget, SELFEVOSKILL attains a 73.9% mean task reward, exceeding the no-skill condition by +33.0 percentage points and the benchmark's curated skill by +17.2 percentage points, while skill evolution receives neither task rewards nor benchmark verifier outputs. In controlled ablations, using two with-skill runs instead of a matched with/without-skill pair reduces mean reward by 4.6 percentage points, while limiting revision to one round reduces it by 9.0 points relative to repeated revision. Together, these results show that behavioral differences between matched executions can provide an effective signal for revising skills when outcome supervision is unavailable.

1 INTRODUCTION

Agent skills equip large language model (LLM) agents with reusable procedures for professional tasks without updating model parameters. Under the Agent Skills specification, a skill is a package of procedural instructions that may also include scripts and reference resources (Anthropic, 2025; Xu & Yan, 2026). Although curated skills improve task completion on professional agent benchmarks (Li et al., 2026), an available skill may not meet the requirements of the current task. For example, a Git workflow skill may instruct an agent to push changes directly to the default branch, even though the repository requires feature branches and pull requests. Following the skill exactly would then violate the repository's workflow requirements. This mismatch motivates studying how to revise existing skills to meet the requirements of the current task.

Existing research on automatic skill improvement includes three relevant lines of work. The first extracts reusable procedures from past execution trajectories (Wang et al., 2024a; Zhao et al., 2024; Shinn et al., 2023). The second revises existing skills and uses validation performance, ground-truth answers, or oracle judgments to decide which changes to retain (Yang et al., 2026a; Zhang et al., 2026b; Alzubi et al., 2026). The third updates skills using failures or feedback collected during deployment (Zhang et al., 2026a; Liu et al., 2026b; Ma et al., 2026). The first line focuses on acquiring reusable procedures rather than revising an existing skill for the current task. The latter two address skill revision, but typically rely on external outcome signals to guide updates or select revisions. Obtaining these signals may require dedicated verifiers, labeled solutions, additional

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

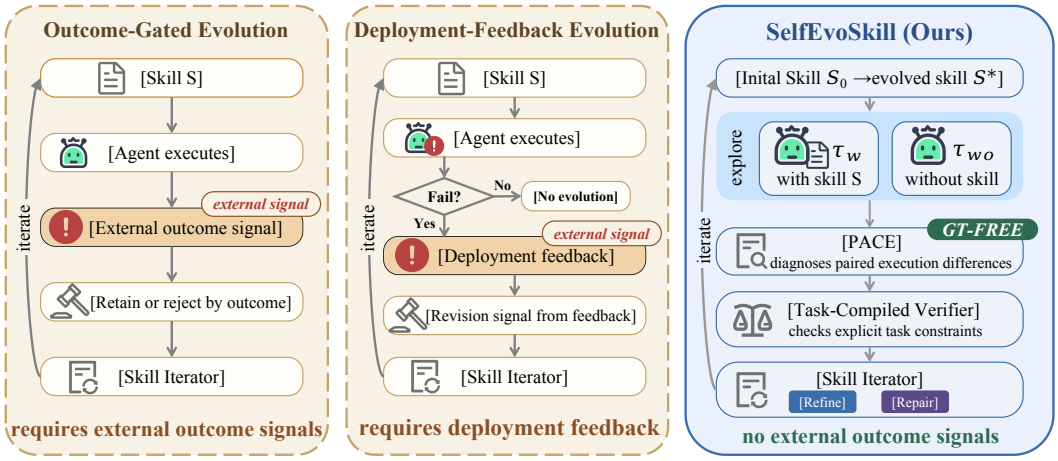


Figure 1: Three paradigms for skill revision. *Outcome-Gated* (left) and *Deployment-Feedback* (center) rely on external outcome signals. SELFEVOSKILL (right) uses behavioral differences between matched executions with and without the skill to locate relevant skill passages and iteratively revise them, without external outcome supervision.

validation tasks, or feedback collected after deployment. These signals may be costly or unavailable for a new task, making it difficult to revise an existing skill using methods that depend on them. This motivates our central question: can an agent revise an existing skill using only the task, workspace, and observable execution traces, without external outcome supervision?

Execution traces provide evidence of how an agent carries out a task, including the instructions it follows, the tools it uses, and the artifacts it produces. However, observing a problematic action in a single execution is not sufficient to determine whether the skill contributed to that action, since the agent may behave similarly without the skill. An execution without the skill therefore provides a useful reference for interpreting the behavior observed when the skill is loaded. Comparing the two executions under the same task, workspace, and agent settings helps distinguish behavior associated with skill loading from behavior that also occurs without it. In the Git example, a direct push observed in both executions provides limited evidence for attributing the action to the skill's push instruction, whereas a direct push observed only when the skill is loaded gives a more specific reason to examine that instruction. These comparisons therefore provide a basis for locating skill instructions that may contribute to problematic behavior, while subsequent executions allow the effects of revisions to be examined.

We propose SELFEVOSKILL, a framework for iterative skill revision that uses behavioral differences between matched executions as its revision signal, without task rewards, benchmark verifier outputs, or other external outcome supervision. It compares executions with and without the skill under the same task, workspace, and agent settings, then maps the observed differences to specific skill passages. This mapping connects skill instructions to their observed effects, guiding which passages to revise and which to preserve. Each revision is evaluated through subsequent executions, whose traces inform whether to retain the changes and where further revision is needed. The framework rejects changes that introduce harmful behavior, while checks derived once from the task's explicit requirements prevent revisions from violating task constraints. By connecting behavioral comparison, targeted editing, and repeated evaluation, SELFEVOSKILL enables skill adaptation using only the task, workspace, and observable execution traces. Figure 1 contrasts this framework with approaches that rely on external outcome supervision.

Our evaluation examines whether skill revision without external outcome supervision improves task performance and whether paired comparison and repeated revision contribute to that improvement. Across 89 containerized tasks in 8 professional domains, SELFEVOSKILL outperforms the no-skill and curated-skill baselines by +33.0 and +17.2 percentage points in mean task reward under the same evaluation budget (Table 1). Ablations show that comparing executions with and without the skill is more effective than using two executions that both load it, supporting the value of an execution without the skill as a reference. Further gains from repeated revision over a single round support

using subsequent execution evidence to refine earlier edits. Analysis by knowledge type examines how these improvements vary with the execution evidence available for revision.

Contributions. (1) **Skill revision from paired runs without outcome supervision.** We propose SELFEVOSKILL, which uses behavioral differences between matched executions with and without a skill to guide revision without task rewards, benchmark verifier outputs, or other outcome signals. (2) **Iterative passage-level revision from execution differences.** We link behavioral differences to passages to revise or preserve, then use subsequent executions to evaluate edits and guide further revision. Harmful edits are rejected, and fixed checks enforce explicit task constraints. (3) **Empirical gains from skill revision without outcome supervision.** Across 89 tasks in 8 professional domains, SELFEVOSKILL outperforms both baselines. Ablations support paired comparison and repeated revision, while analysis by knowledge type relates gains to available execution evidence.

2 RELATED WORK

Procedural memory and skill construction. Frozen agents can load external procedural knowledge at inference time without updating weights (Yao et al., 2023). Voyager, ExpeL, Reflexion, Self-Refine, and Agent Workflow Memory distill reusable procedures from execution history or model feedback (Wang et al., 2024a; Zhao et al., 2024; Shinn et al., 2023; Madaan et al., 2023; Wang et al., 2024b). Agent Skills specify this knowledge as loadable packages (Anthropic, 2025; Xu & Yan, 2026); Trace2Skill, SkillWeaver, AutoSkill, and SkillFoundry construct such packages from trajectories or other experience (Ni et al., 2026; Zheng et al., 2025; Yang et al., 2026b; Shen et al., 2026). SkillTTA further synthesizes a temporary skill from retrieved task-relevant trajectories at test time (Wang et al., 2026b). The memory and synthesis methods above create, retrieve, or accumulate procedural knowledge; they do not isolate the incremental effect of an already available skill in a matched execution pair. We instead ask how to locally revise such a skill for the task at hand when neither task reward nor a held-out validation result is available.

Skill revision with outcome supervision. Several lines of work iteratively revise skills but retain updates using an external outcome signal (see Figure 1). SkillOpt, EvoSkill, CoEvoSkills, SkillsVote, and SkillRevise use held-out validation performance, ground-truth answers, oracle judgments, success-gated evidence, or re-executed empirical utility to select updates (Yang et al., 2026a; Alzubi et al., 2026; Zhang et al., 2026b; Liu et al., 2026a;c). SkillForge, SkillClaw, and SkillEvolver revise skills from deployment failures or feedback (Liu et al., 2026b; Ma et al., 2026; Zhang et al., 2026a); RL-based approaches optimize task reward directly (Wang et al., 2025; Xia et al., 2026; Vishe et al., 2026; Wang et al., 2026a; Shi et al., 2026; Ouyang et al., 2026; Zhang et al., 2026c). These methods require external outcome signals unavailable in our setting.

3 METHOD

3.1 PROBLEM FORMULATION

Given a natural-language task description T, an initial workspace state W, an initial skill S_0, and a frozen agent π, our goal is to revise S_0 to improve task performance without updating the agent's parameters. For a candidate skill S, execution produces a trajectory $\tau \sim \pi(\cdot \mid S, T, W)$. Let $\mathcal{R}(\tau) \in [0, 1]$ denote the task reward. The ideal objective is

$$S^* \in \arg\max_S \mathbb{E}_{\tau \sim \pi(\cdot|S,T,W)} \left[\mathcal{R}(\tau)\right]. \tag{1}$$

We study this objective without external outcome supervision, under the information restriction $\mathcal{C}_{\text{osf}}$. The revision procedure may access T, W, the initial and candidate skills, and observable execution traces and artifacts. It may not access $\mathcal{R}$ or its outputs, held-out validation results, reference solutions, benchmark verifier outputs, or oracle pass/fail feedback. Checks derived solely from explicit task requirements and observable workspace information are permitted. Thus, $\mathcal{R}$ defines the performance objective but provides no feedback during revision.

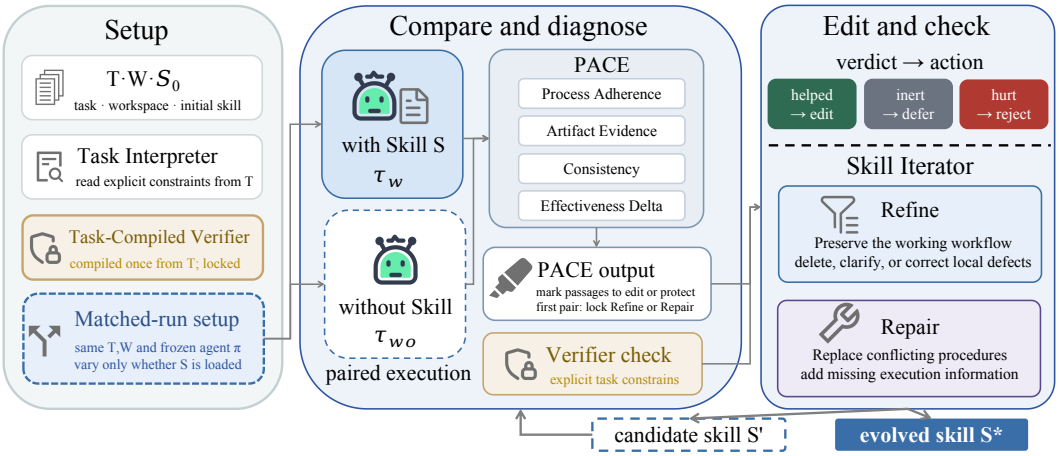


Figure 2: SELFEVOSKILL overview. *Setup* (left): compile a verifier from explicit task requirements and select *Refine* or *Repair* through an initial paired audit. Both remain fixed. *Compare and diagnose* (center): PACE compares executions with and without the skill to identify passages to edit or preserve. *Edit and check* (right): the Skill Iterator edits the skill under the selected mode, and subsequent execution checks the candidate. A *hurt* verdict or constraint violation triggers rejection.

3.2 SELFEVOSKILL OVERVIEW

SELFEVOSKILL revises skills through an iterative process of execution comparison, targeted editing, and evaluation, without external outcome supervision. It combines behavioral evidence from matched executions with checks derived from explicit task requirements: the former guides skill revision, while the latter constrains the resulting changes.

At each iteration, the agent executes the task with and without the current skill under the same task, initial workspace state, and agent settings. PACE, our trajectory comparison and skill diagnosis module, examines the paired trajectories and the current skill along four dimensions: Process Adherence, Artifact Evidence, Consistency, and Effectiveness Delta. It links observed behavioral differences to skill passages that should be revised or preserved. The Skill Iterator then applies targeted edits under Refine or Repair, a revision mode selected by the initial paired audit and fixed throughout the loop. Subsequent executions assess the updated skill and provide evidence for further revision. The Task-Compiled Verifier, constructed before revision begins and held fixed throughout the loop, checks compliance with explicit task requirements.

This process allows revision decisions to be reconsidered as edits change the agent's behavior. Figure 2 illustrates the workflow. The following subsections describe *paired trajectory auditing*, *PACE*, *refine and repair*, and *the task-compiled verifier*.

3.3 PAIRED TRAJECTORY AUDITING

SELFEVOSKILL examines the influence of the current skill through two executions of the frozen agent π: one with S loaded and one without a skill. The resulting trajectories are $\tau_w \sim \pi(\cdot \mid S, T, W)$ and $\tau_{wo} \sim \pi(\cdot \mid \varnothing, T, W)$, where $\varnothing$ denotes the absence of a skill. Both executions start from the same workspace state W and use the same task T and agent configuration. Skill loading is the only factor deliberately varied. The execution without the skill provides a reference for assessing which observed behaviors may be associated with the skill.

The audit compares observable tool calls, decisions, and generated artifacts, without accessing the task reward $\mathcal{R}$. If the execution with the skill satisfies an explicit task requirement that the execution without it does not, the difference provides evidence of a beneficial skill effect on that requirement. Matching behavior provides no evidence of an incremental effect for the behavior being compared. Conversely, a call or artifact that conflicts with T indicates a problem in execution, whose connection to the skill must be examined using the paired trajectories and skill instructions. PACE performs this diagnosis and identifies its implications for revision. If the execution without the skill is too

incoherent to provide a meaningful reference, the framework disables edits that import fragments from that trajectory. It relies on checks of the execution with the skill until a later pair provides an informative comparison (Appendix A).

3.4 PACE

PACE translates paired execution evidence into passage-level diagnoses. It examines τ_w, τ_{wo}, and the current skill S together, since a behavioral difference alone does not identify the instruction associated with it. Its outputs comprise a behavioral verdict and specific skill passages marked for revision or preservation. These passage-level recommendations guide the Skill Iterator.

PACE evaluates four complementary dimensions: **(1) Process Adherence.** This dimension examines whether the agent follows the steps and tool-use patterns specified in S, providing evidence about which instructions are enacted during execution. **(2) Artifact Evidence.** This dimension checks the files produced by both runs against explicit requirements in T and the deterministic checks provided by the Task-Compiled Verifier. The skill may be used to investigate an observed mismatch, but never to define the requirements against which artifacts are assessed. **(3) Consistency.** This dimension examines whether the workflow specified in S is compatible with the task's data schema and output format. **(4) Effectiveness Delta.** This dimension assesses the incremental behavioral contribution of loading S relative to the execution without it. Observable calls, decisions, and artifacts support a judgment of clearly positive, near-zero, or negative contribution. This judgment concerns observed execution behavior, not task reward or estimated task success.

All four dimensions use only the task, permitted workspace information, skill instructions, and observable execution evidence. None accesses task rewards, benchmark tests, benchmark verifier outputs, reference solutions, or oracle pass/fail feedback. See Appendix D for evaluator details.

Verdict aggregation and use. PACE aggregates the judgments according to a precedence rule. Any individual *hurt* judgment takes precedence. Otherwise, Artifact Evidence determines the verdict when the file outputs resolve the comparison. If they do not, a majority of the remaining judgments determines a *helped* or *inert* verdict. The verdict summarizes the behavioral evidence, while the passage-level diagnosis identifies the corresponding revision or preservation actions. For the initial skill, Effectiveness Delta and the scope of the diagnosis determine the fixed Refine and Repair modes (§3.5). For an edited candidate, a *hurt* verdict triggers rejection.

3.5 Refine and Repair Modes

SelfEvoSkill distinguishes between preserving a useful procedure and replacing an ineffective or conflicting one. The initial paired audit selects either *Refine* or *Repair*, and the selected mode remains fixed throughout revision.

Mode selection. A clearly positive Effectiveness Delta for S_0 selects Refine; a negative one selects Repair. For near-zero judgments, local redundancy, ambiguity, or clarifiable defects select Refine, while missing execution information, nonlocal workflow conflicts, or harmful procedures select Repair. The mode remains fixed thereafter.

Permitted edits. The Skill Iterator edits passages identified by PACE. Refine preserves the executed procedure while removing unused passages, clarifying instructions, or correcting local defects. Repair permits removing conflicting procedures, replacing them with fragments from τ_{wo} checked for compliance with T, and then adding missing execution details. Both modes preserve protected passages. Repair allows all passages to be examined but respects these protections. The candidate is evaluated in the next execution with the skill.

Initial diagnosis and candidate rejection. Mode selection is distinct from candidate acceptance. The initial audit fixes the mode even when the PACE verdict is *inert*, and a negative initial diagnosis can select Repair rather than terminate revision. Once an edit produces a candidate, however, a subsequent *hurt* verdict rejects that candidate and restores the most recent skill that passed the verifier. This distinction allows an initially ineffective or misleading skill to be repaired while preventing harmful candidates from replacing an earlier skill.

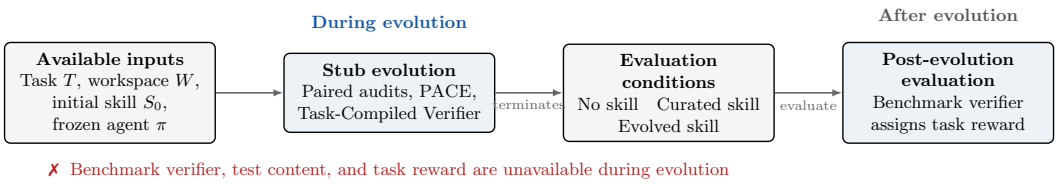


Figure 3: Evaluation protocol without outcome supervision. Evolution uses task inputs and execution traces without access to benchmark verifiers, tests, or rewards. After evolution, the same agent is evaluated under no-skill, curated-skill, and evolved-skill conditions using the benchmark verifier.

3.6 The Task-Compiled Verifier

Improvement in the paired comparison does not guarantee compliance with explicit task requirements. Before revision begins, SelfEvoSkill compiles requirements in T that admit deterministic checks into a Task-Compiled Verifier. Workspace data W is used only to resolve task-referenced files and recompute values required by T, not to introduce additional requirements. Checks cover required files and paths, headers or schemas, enumerated value sets, recomputable values, and companion files. Compilation excludes the initial skill, paired trajectories, task rewards, benchmark tests, and benchmark verifier outputs. The verifier remains fixed throughout revision.

The verifier checks each candidate through its subsequent execution with the skill. A detected constraint violation rejects the candidate and restores the most recently accepted skill, regardless of the PACE verdict. If S_0 fails, the failure is recorded because no earlier skill is available for rollback. When the initial audit selects Repair, subsequent edits may use the execution without the skill as a reference. The verifier enforces explicit task constraints. It neither assesses the skill's behavioral contribution nor selects the revision mode, and it does not replace benchmark evaluation. See Appendices C and A for the full algorithm and implementation details.

4 Experiments

4.1 Experimental Settings

Benchmark and metric. We evaluate on a fixed snapshot of SkillsBench containing 89 tasks across 8 professional domains (Li et al., 2026). Each task provides a curated skill, a Harbor container (Merrill et al., 2026), and a deterministic benchmark verifier that returns a task reward in $[0, 1]$, our evaluation metric. See Appendix F for task details and results.

Revision setup. Skill revision takes place in a stub container without access to the benchmark's pytest verifier or test content. The benchmark verifier is invoked only after revision terminates. Agent execution, PACE assessment, skill editing, and evaluation runs all use Claude Code with Claude Opus 4.8. Revision allows at most five paired audits per task. All tasks use the same twelve PACE evaluator templates, paper-mode prompts, and fixed rule for selecting the revision mode, with no task-specific tuning. Figure 3 illustrates the separation between revision and evaluation.

Baselines and evaluation protocol. We compare three conditions using the same frozen agent: no skill, the benchmark's curated skill, and the skill revised by SelfEvoSkill. The first two conditions each use two independent task executions. For SelfEvoSkill, two independent revision runs produce two skills, each evaluated once. We report the higher of the two rewards for each condition, following a common best-of-two reporting protocol.

4.2 Main Results

Table 1 reports per-domain mean task reward, and Table 2 compares SelfEvoSkill with other skill-evolution methods on SkillsBench.

Paired auditing without outcome supervision improves task reward. Over the 89 tasks, SelfEvoSkill attains 73.9%, with gains of +33.0 pp over the no-skill condition and +17.2 pp over the curated skill. The evolved skill outperforms the curated skill in seven of eight domains.

Table 1: Mean task reward (%) on the fixed 89-task snapshot across 8 domains, evaluated with Claude Opus 4.8. The three performance columns report mean per-task rewards in $[0, 1]$, expressed as percentages. Curated skill denotes the benchmark-provided skill supplied with the task; SELFEVOSKILL denotes the evolved skill. In the **Overall** row, N gives the total task count, and the performance entries are computed over all 89 tasks rather than as an unweighted mean of the eight domain rows. The last column reports the gain of SELFEVOSKILL over the curated skill.

Domain	N	No skill	Curated skill	SELFEVOSKILL	Δ vs Curated skill (pp)
Software Engineering	16	53.8	40.3	**78.8**	+38.5
Office & White Collar	15	60.0	60.0	**86.7**	+26.7
Natural Science	15	41.6	73.3	**83.7**	+10.4
Industrial & Physical Sys.	14	21.4	61.2	**71.4**	+10.2
Finance & Economics	9	33.3	**44.4**	**44.4**	+0.0
Mathematics & OR	8	25.0	35.6	**57.5**	+21.9
Cybersecurity	6	41.7	63.8	**66.7**	+2.9
Media & Content Prod.	6	33.8	79.2	**83.3**	+4.1
Overall	**89**	40.9	56.7	**73.9**	**+17.2**

In Finance & Economics, the evolved and curated skills both attain 44.4%, so this condition is a tie rather than an improvement. The largest gains occur in Software Engineering (+38.5 pp), Office & White Collar (+26.7 pp), and Mathematics & OR (+21.9 pp). In Software Engineering, the curated skill is below the no-skill condition. In Office & White Collar, it provides no gain over the no-skill condition. Mathematics & OR has the lowest curated-skill reward among the eight domains. These conditions leave substantial room for skill revision. (Appendix F).

Table 2: Comparison with other methods on SkillsBench. External results are taken from the cited papers and were not reproduced in our harness. The values do not constitute a controlled comparison.

Method	Executor	Reward (%)	External outcome signal
SELFEVOSKILL	Opus 4.8	73.9	None
CoEvoSkills	Opus 4.6	71.1	Oracle
SkillRevise	Opus 4.7	51.2	Verifier

The evolved skill achieves comparable reward without outcome supervision. Table 2 compares SELFEVOSKILL with two other skill-evolution methods evaluated on SkillsBench: CoEvoSkills (Zhang et al., 2026b) and SkillRevise (Liu et al., 2026c). Despite receiving no external outcome signals during evolution, SELFEVOSKILL reports a mean reward in the same performance range as these outcome-supervised methods. Table 4 details the differences in task snapshots, models, harnesses, supervision signals, and reporting protocols.

4.3 ABLATION STUDIES

We ablate four design choices in SELFEVOSKILL: the execution without the skill as a reference, audit-based selection between Refine and Repair, the addition of missing execution information during Repair, and iterative revision with PACE. All variants use GLM-5.2 as the evolution model and share the same task set and evaluation protocol, allowing controlled comparisons of these design choices. The full method achieves a mean reward of 73.0% under GLM-5.2, providing evidence of effectiveness with a second evolution backbone alongside the 73.9% result under Claude Opus 4.8.

The without-skill reference provides the update signal. If both executions load the same skill, trajectory differences reflect sampling variation and cannot indicate whether the skill is used. We hold the procedure fixed and change only the pairing. Both conditions use two executions per iteration, matching the compute budget: one run with the skill and one without, versus two runs that both load the same skill. The comparison follows the same best-of-two reporting protocol. Both arms are evolved and evaluated with GLM-5.2 on all 89 tasks, rather than the Claude Opus 4.8 protocol used for Table 1. Mean reward is 64.0% for the with/without pair and 59.4% for two with-skill runs (+4.6 pp). The without-skill run provides a contrastive reference for estimating the skill's incremental effect.

Table 3: Ablation on 89 tasks. **(a)** Mean reward (%) by variant; **(b)** complementary solves (%) by fixed edit mode. All skills evolve with GLM-5.2; missing or failed runs score 0.

Variant	GLM	Qwen	DS
Refine-only	62.9	46.1	38.2
Repair-only	67.0	50.4	39.7
Repair, replace only	65.1	44.7	37.4
Single-round PACE	64.0	42.7	35.4
Adaptive routing	**73.0**	**53.2**	**42.4**
Per-task max (upper bound)	77.3	58.5	47.9

(a) Mean reward by variant.

Category	GLM	Qwen	DS
Both solve	52.8	39.3	29.2
Refine-only solves	10.1	6.7	9.0
Repair-only solves	13.5	11.2	10.1
Neither solves	23.6	42.7	51.7

(b) Complementary solves.

A paired reference prevents an accidental trace from triggering an edit. The aggregate comparison above establishes a difference in mean reward; `grid-dispatch-operator` records the decision behind it. With the without-skill reference, the framework observes that the with-skill workflow remains optimal while the without-skill run fails load balance, so it leaves the skill unchanged. When both runs load the skill, a silent infeasibility is judged as skill harm; the resulting rewrite still passes structural checks but receives task reward 0. This archived example illustrates how the paired reference guides the update in the GLM pairing run (Appendix E.6).

Adaptive routing preserves the complementary strengths of both modes. A workflow that improves execution should be preserved, whereas one with no effect or that misleads the agent may be replaced. If only one edit mode is permitted in both cases, some tasks become unrecoverable. In Table 3 and Figure 4a, Refine-only attains 62.9% on GLM and Repair-only attains 67.0%. Table 3b shows that Refine uniquely solves 10.1% of tasks and Repair uniquely solves 13.5%: using only one mode forfeits the tasks that the other mode solves. Selecting the mode from the first paired audit reaches 73.0% on GLM, near the per-task upper bound of 77.3%. When these GLM-evolved skills are evaluated on Qwen and DeepSeek without re-evolution, they reach 53.2% and 42.4%, still above either mode alone (§3.5).

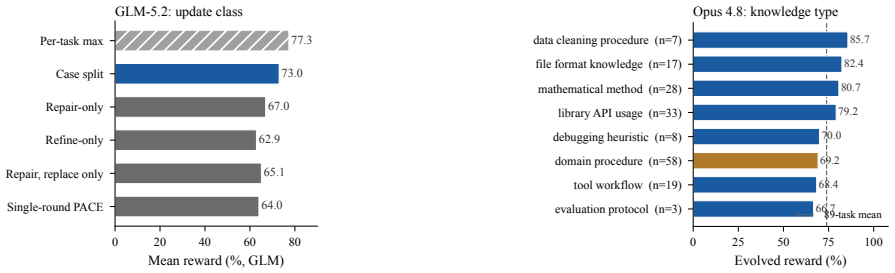
Repair benefits from adding missing execution information. After removing conflicting passages, Repair can add missing execution information. Allowing this operation improves the GLM reward from 65.1% under replacement only to 67.0% under Repair. Thus, replacement alone does not recover all execution information used by Repair.

Repeated audits expose additional edit opportunities. Restricting PACE to a single round attains 64.0% on GLM, 9.0 pp below the multi-round variant with the selected edit mode (73.0%). A single audit round can leave actionable differences unaddressed; later rounds act after an earlier edit changes the execution trace.

4.4 Analysis of Paired Process Evidence

Skills with observable execution signals benefit most from paired revision. Skills whose procedures specify library functions or mathematical formulas exceed the most frequent domain-procedure label (Figure 4b): library API usage attains 79.2% ($n = 33$) and mathematical method attains 80.7% ($n = 28$), compared with 69.2% for domain procedure ($n = 58$). For these skills, execution often exposes runtime failures or incorrect values, providing concrete signals that make relevant passages easier to diagnose. To obtain these per-type statistics, we constructed an eight-category taxonomy from the associated skills and task set. An LLM assigned one or more labels from this taxonomy to 89 tasks, and two authors reviewed and corrected every assignment. Labels overlap, and n denotes the number of tasks bearing a label. Per-type statistics appear in Appendix F.

A case study identifies a conflicting instruction. In `paper-anonymizer`, the loaded skill aborts a redaction unless more than 80% of the source text remains, and its verification code enforces a 0.7 floor, although the task specification requires only that at least 60% (0.6) be retained. The over-strict guard therefore rejects valid redactions. PACE marks that guard as an edit target (ST-002/ST-003) and unwraps a generic function wrapper (ST-001), protecting the remaining sections;



(a) Routing from the first paired audit.

(b) Mean evolved reward by knowledge type.

Figure 4: Paired routing and performance by knowledge type. **(a)** Audit-based mode selection outperforms either fixed mode in the GLM ablation; hatched bars show the per-task upper bound. **(b)** Mean rewards by knowledge type on the Opus 89-task split. Labels may overlap; gold denotes domain procedure, and the dashed line marks the overall mean.

the repaired skill sets both thresholds to 0.6, and the task's evolved-skill reward rises from 0 to 1 (Repair). This case records a passage-level diagnosis for a conflicting instruction (Appendix E.5).

Two case studies trace protection and repair decisions. The following recorded cases trace two distinct decisions. In `software-dependency-audit`, Refine removes only unused or conflicting passages from a workflow supported by the paired runs: it removes tutorial prose and hardcoded paths, cutting about 35% of lines, and the evolved reward is 1.0. In `data-to-d3`, the first with-skill run passes 12 of 13 checks but violates the task's exact column-label requirement, whereas the without-skill run passes all 13. The locked verifier prevents that failing state from being accepted, while the Repair mode selected by the same audit remains available. By the final archived iteration, the evolved skill passes all 13 checks after shrinking from 188 lines to 50, and its reward rises from 0 to 1.0. This case illustrates the verifier's role within a repair trajectory. Appendix E gives complete records, and Appendix E.7 reports the resulting package-level changes.

Failure analysis identifies execution as the largest residual bottleneck. We analyzed 25 non-passing cases using their final evaluation outputs, evolved skills, and audit records. In 14 cases (56.0%), the evolved skill contained the relevant task guidance, but the frozen agent failed to follow it correctly in the final output. Five cases (20.0%) resulted from incorrect or harmful skill revisions, four (16.0%) from PACE failing to localize an effective edit, and two (8.0%) remained uncertain because the available records did not support a unique attribution. Execution was therefore the largest observed failure source, while nine cases directly point to opportunities for improving passage localization and skill editing.

Limitations of paired process evidence. Execution errors by the frozen agent account for the largest share of non-passing cases (56.0%): paired process evidence can identify passage-level defects in the skill but cannot compensate when the agent fails to follow a correctly revised procedure. The remaining nine attributable cases—incorrect revisions (20.0%) and localization failures (16.0%)—point to opportunities for improving how trajectory differences are connected to skill passages and how edits are applied. Appendix F provides the task-level breakdowns.

5 Conclusion

We present SelfEvoSkill, which uses behavioral differences between matched executions with and without a skill to guide iterative passage-level revision without external outcome supervision. A fixed verifier enforces explicit task requirements. Across 89 tasks in 8 professional domains, SelfEvoSkill achieves a mean reward of 73.9%, outperforming both baselines. Ablations support paired comparison and repeated revision, showing the value of execution evidence for skill adaptation without external outcome supervision. This approach offers a practical way to adapt existing skills when task rewards and reference solutions are unavailable.

AI USE STATEMENT

We used generative AI tools for two purposes in preparing this submission: polishing the English wording and grammar of the manuscript and assisting with scripts for experiments and data processing. In these supporting roles, generative AI tools were not used to generate research ideas, design the method or evaluation protocol, or produce, select, or interpret any reported numerical results. All AI-assisted text was reviewed and edited by the authors and checked for consistency with the underlying results. All AI-assisted code was reviewed and tested by at least two authors, who also inspected the recorded execution logs to check its behavior. Generative models are also used within the framework under study, serving as the frozen agent and the PACE evaluators. This use is part of the method and is described in §3 and Appendix D. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects and collects no personally identifiable information. Every execution runs inside a containerized sandbox on the benchmark's own workspace data, and the models and the benchmark we use are employed under their respective licenses and intended-use terms. One risk is specific to our setting: because SELFEVOSKILL rewrites procedural instructions that a frozen agent then executes, an evolved skill could in principle encode an unsafe procedure. Two mechanisms address this. PACE includes a safety-compliance evaluator that vetoes an update on a critical violation, such as unsafe paths, exposed credentials, or destruction of content the task requires to be preserved (Appendix D); and the Task-Compiled Verifier rejects updates that violate explicit task-grounded constraints. Neither mechanism substitutes for human review before an evolved skill is deployed outside a sandbox. The authors declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

The problem setting and the constraint on revision without outcome supervision are stated formally in §3.1. The evolution loop, including the convergence criteria and locked initial routing rule, is given as pseudocode in Appendix C; the complete fixed PACE evaluator inventory is in Appendix D; and implementation details, including model versions, the iteration and edit budgets, and the task-compiled verifier boundary, are in Appendix A. The frozen task manifest and per-task rewards in Appendix F identify the evaluation snapshot and support recomputation of Table 1. Six worked cases in Appendix E give the per-iteration decisions and the resulting skill edits. We will release the implementation, paper profile, paper-mode prompts, static evaluator templates, frozen manifest, selection and budget rules, launch scripts, audit-record schema, and evolved skill packages.

REFERENCES

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, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457. Oral.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

Anthropic. Agent skills. https://code.claude.com/docs/en/skills, 2025.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions, 2025. URL https://arxiv.org/abs/2502.10325.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei

Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023. URL https://arxiv.org/abs/2310.03714.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, 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, 2026. URL https://arxiv.org/abs/2602.12670.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.20050.

Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, Yuyu Luo, and Zhiyu Li. Skillsvote: Lifecycle governance of agent skills from collection, recommendation to evolution, 2026a. URL https://arxiv.org/abs/2605.18401.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support, 2026b. URL https://arxiv.org/abs/2604.08618.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, Xiyu Ren, Ruan Chenyu, Haoran Li, and Yangqiu Song. Skillrevise: Improving LLM-authored agent skills via trace-conditioned skill revision, 2026c. URL https://arxiv.org/abs/2606.01139.

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, 2026. URL https://arxiv.org/abs/2604.08377.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

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, 2026. URL https://arxiv.org/abs/2605.06614.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources, 2026. URL https://arxiv.org/abs/2604.03964.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning, 2026. URL https://arxiv.org/abs/2605.06130.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Yash Vishe, Rohan Surana, Xunyi Jiang, Zihan Huang, Xintong Li, Nikki Lijing Kuang, Tong Yu, Ryan A. Rossi, Jingbo Shang, Julian McAuley, and Junda Wu. Skill-r1: Agent skill evolution via reinforcement learning, 2026. URL https://arxiv.org/abs/2605.09359.

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*, 2024a.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents, 2026a. URL https://arxiv.org/abs/2604.10674.

Jingxing Wang, Chenyu Zhou, Zhihui Fu, Jun Wang, Weiwen Liu, Weinan Zhang, and Jianghao Lin. Skills on the fly: Test-time adaptive skill synthesis for LLM agents, 2026b. URL https://arxiv.org/abs/2605.16986.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library, 2025. URL https://arxiv.org/abs/2512.17102.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024b. URL https://arxiv.org/abs/2409.07429.

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, 2026. URL https://arxiv.org/abs/2602.08234.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward, 2026. URL https://arxiv.org/abs/2602.12430.

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, 2026a. URL https://arxiv.org/abs/2605.23904.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. Autoskill: Experience-driven lifelong learning via skill self-evolution, 2026b. URL https://arxiv.org/abs/2603.01145.

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

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text, 2024. URL https://arxiv.org/abs/2406.07496.

Genrui Zhang, Erle Zhu, Jinfeng Zhou, Caiyan Jia, and Hongning Wang. Skillevolver: Skill learning as a meta-skill, 2026a. URL https://arxiv.org/abs/2605.10500.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026b. URL https://arxiv.org/abs/2604.01687.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents, 2026c. URL https://arxiv.org/abs/2602.02474.

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 (ICLR)*, 2026d. URL https://arxiv.org/abs/2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2024.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills, 2025. URL https://arxiv.org/abs/2504.07079.

Appendix

A. Implementation Details 15
B. Related Work 17
C. Evolution Loop Algorithm 18
D. PACE Evaluator Inventory 19
E. Supplementary Evidence and Case Studies 22
E.1 Case A: `software-dependency-audit` (Refine, primary) 22
E.2 Case B: `data-to-d3` (Repair, primary) 23
E.3 Case C: `lab-unit-harmonization` (Refine) 23
E.4 Case D: `exceltable-in-ppt` (Repair) 24
E.5 Case E: `paper-anonymizer` (Repair; passage-level diagnosis) 24
E.6 Case F: `grid-dispatch-operator` (Refine; pairing contrast) 25
E.7 Structural Analysis Details 25
F. Task Manifest, Per-Task Results, and Label Statistics 26

A Implementation Details

Domain consolidation. The frozen snapshot's 11 fine-grained domains are consolidated into 8 by merging small domains and renaming one: *Healthcare* into *Natural Science*; *Energy*, *Manufacturing*, and *Robotics* into *Industrial & Physical Systems*; and *Mathematics* renamed to *Mathematics & OR*.

Reporting two independent runs. For each task, we conduct two independent evolution runs, yielding two evolved skills that are each evaluated once by the benchmark verifier after evolution terminates. We report the higher of these two task rewards, following the fixed best-of-two protocol in §4.1. A full evolution run is lengthy and can be affected by stochastic agent execution and transient API or connection failures. Using two prespecified runs reduces sensitivity to these factors. Missing, timed-out, or aborted runs receive a reward of zero. No additional runs are launched after observing benchmark rewards.

Protocol differences in the cross-paper comparison. Because rerunning CoEvoSkills and SkillRevise under a common harness exceeds our available compute budget, Table 2 uses the results reported in their respective papers. Table 4 summarizes the differences relevant to interpreting those values. The task snapshots, models, harnesses, supervision signals, and across-run aggregation differ, so the comparison provides contextual positioning rather than a controlled ranking.

Table 4: Protocol differences behind the contextual comparison in Table 2. CoEvoSkills describes an 87-task evaluation set and reports evolution statistics for 86 tasks. SkillRevise reports 44 successes among 86 SkillsBench tasks under the Opus 4.7 executor. Across-run reporting is distinct from candidate selection within an evolution or revision run.

Dimension	SelfEvoSkill	CoEvoSkills	SkillRevise
Task snapshot	89 tasks	87 evaluated; evolution statistics for 86	86 tasks
Evolution / revision model	Opus 4.8	Opus 4.6	GPT-5.5
Executor / harness	Opus 4.8 / Claude Code	Opus 4.6 / Claude Code	Opus 4.7 / Claude Code
Evolution signal	No external outcome signal	Surrogate-verifier feedback and oracle pass/fail	Verifier-derived utility
Across-run reporting	Per-task best of two independent evolution runs	Mean over five independent runs	One reported aggregate (44/86); no repeated-run aggregation reported
Within-run candidate selection	PACE-guided editing with verifier rollback	Iterative co-evolutionary verification	Best-observed selection over up to three revision rounds

Verdict aggregation. A single *skill_hurt* report from any evaluator vetoes the update (§3.4). Absent any hurt signal, the Artifact Evidence evaluator (*eval-output-evidence-check*) is decisive when task-grounded file checks resolve the comparison; skill text may support attribution but never supplies the checked requirement. Any remaining disagreement is settled by majority vote into *skill_helped* or *skill_inert*.

First paired audit. The first paired audit of S_0 determines which edit mode is permitted (§3.5) using the Effectiveness Delta dimension of PACE. The same traces constitute the first iteration's diagnosis. Subsequent iterations do not recompute the edit mode. A *skill_hurt* verdict vetoes the current update and does not change the locked mode. Effectiveness Delta returns a categorical judgment: clearly positive, near zero, or negative. A clearly positive judgment selects Refine and a negative judgment selects Repair. For a near-zero judgment, local redundancy, ambiguity, or a clarifiable defect selects Refine, whereas missing execution information, a nonlocal workflow conflict, or a harmful procedure selects Repair. This edit-mode choice is derived from paired process evidence, not from $\mathcal{R}$ or from the task reward of the initial skill.

Task-Compiled Verifier generation. Compilation precedes all paired executions and proceeds in two stages. First, an LLM-based task interpreter maps each explicit requirement in T to a structured specification and classifies it as an output-structure check, an invariant, a value recomputable from W, a required companion artifact, or not programmatically verifiable. Workspace data supplies only paths and values referenced by T and contributes no new requirement. Second, an LLM-based compiler maps every eligible requirement to a named deterministic check. Expected values are read or recomputed from W at verification time; hard-coded answers are prohibited. A requirement that cannot be traced to T and checked without a reference solution is excluded rather than approximated.

Before paired evolution begins, the generated verifier must fail on an empty output directory, emit a structured result for every admitted requirement, and record the constraint map and requirement identifiers. A verifier that passes these validation steps is locked and reused unchanged throughout evolution; a run without a valid locked verifier does not proceed. No task-specific verifier is manually modified. Initial-skill text, paired traces, evaluator reports, task rewards, benchmark tests, reference solutions, and benchmark verifier outputs are excluded from compilation. This conservative admission rule makes a failed check evidence of a violated explicit requirement rather than a judgment of overall task quality.

Skill Iterator edit operations. Given the `surgery_targets` and `protected_segments` from PACE together with the paired trajectory evidence, the Skill Iterator applies edits in a fixed priority order: (i) remove or correct any passage flagged as harmful; (ii) for Refine, delete or clarify redundant or conflicting passages and correct local defects, without replacing the prescribed workflow; (iii) for Repair, replace a harmful passage with a verified alternative extracted from τ_{wo}, then supply missing execution information. Every edit must target a specific surgery target and be grounded in the paired-trajectory difference, and protected segments are never altered. Relative to a baseline of B lines, Refine may add at most $0.1B$ net lines per round and Repair at most $0.2B$; in both cases the package is capped at $\max(1.3B, B + 300)$ lines within five iterations. After each commit the change is versioned with git, so any edit that later produces a *skill_hurt* verdict or a Task-Compiled Verifier failure can be rolled back to the previous committed state.

Overlapping edit conflicts. A surgery target and a protected segment can overlap, for example a code block that mixes a removable tutorial comment with an API call required by τ_w. The Skill Iterator resolves such conflicts conservatively: protection takes precedence at the finest available granularity. The overlapping span is split at the protected boundary, only the unprotected remainder is eligible for deletion or replacement, and if a clean split is not possible the entire span is retained. This biases the framework toward under-editing rather than destroying passages required by τ_w, consistent with the asymmetric cost of a harmful update.

Degenerate without-skill runs. When the without-skill trajectory τ_{wo} fails to provide a coherent reference (§3.3), the Effectiveness Delta evaluators record a null incremental signal rather than a fabricated comparison, the swap protocol (which would import a verified snippet from τ_{wo}) is disabled, and the iteration relies on the Task-Compiled Verifier and the single-trajectory evaluators applied to τ_w. The skill can still be pruned or corrected, but supplying missing execution information from the contrast is suspended until a future iteration yields an informative pair.

Protocol record and reproducibility. All model-mediated operations in the main experiments—including paired task execution, PACE auditing, Task-Compiled Verifier generation, and skill editing—use Claude Code 2.1.158 with Claude Opus 4.8. We set the temperature to 0.1 and cap the context window at 1M tokens. Each evolution execution is limited to two hours and at most five iterations, while external benchmark evaluation is limited to one hour. Ablation variants evolve skills with GLM-5.2 and evaluate under GLM-5.2, Qwen3.8-Max, and DeepSeek-V4-Pro (§4.3). For each task, the initial audit records the matched-pair paths, categorical Effectiveness Delta, diagnosis scope, surgery targets, protected passages, locked edit mode, verifier reference, and profile fingerprint; the run record retains subsequent edit and line-count changes. These records exclude task rewards, benchmark tests, reference solutions, and benchmark verifier outputs. The released artifact will include the paper profile, controller and role prompts, the fixed template inventory, the verifier-generation specification and generation records, selection and budget rules, launch scripts, the audit-record schema, the frozen manifest, and evolved skill packages for all 89 tasks.

Anonymous code availability. An anonymized implementation is available at `https://anonymous.4open.science/r/self-evo-skill-0F24/`.

Table 5: Paper-mode protocol and initial-audit record. The protocol fixes the information boundary and the first routing decision without exposing benchmark verifier outputs during evolution.

Component	Contract
Task-Compiled Verifier	Maps explicit deterministic constraints in T to checks, using W only for task-referenced data; validates and locks the verifier before paired evolution; excludes skills, traces, evaluator reports, task rewards, benchmark tests, reference solutions, and benchmark verifier outputs; receives no manual task-specific edits.
PACE inventory	Applies the same twelve templates to every paired audit; a template without task-specific evidence emits no signal rather than being replaced or routed by an external outcome signal.
Initial audit	Records the paired traces, Effectiveness Delta, diagnosis scope, surgery targets, protected passages, verifier reference, profile fingerprint, and a locked Refine or Repair mode.
Edit policy	Permits at most five iterations; Refine and Repair add at most $0.1B$ and $0.2B$ net lines per round, respectively, with a common cap of $\max(1.3B, B + 300)$.

B Related Work

Adjacent instruction optimization and paired execution. DSPy, GEPA, and related prompt or context-playbook methods optimize prompts against an evaluation function or execution success (Khattab et al., 2023; Agrawal et al., 2026; Zhang et al., 2026d; Yuksekgonul et al., 2024). Process-supervision methods in reasoning instead train on step-level labels or process reward models (Lightman et al., 2024; Choudhury, 2025). Anthropic's skill-creator is the closest comparison, because it also compares executions with and without a skill, but practitioner-supplied test assertions still decide whether to retain an edit (Anthropic, 2025). In contrast, SelfEvoSkill uses the behavioral differences between the two executions to select skill passages for revision and a Task-Compiled Verifier to reject violations of explicit task constraints, without task rewards, held-out validation results, or practitioner-supplied test assertions.

Table 6: Revision-method properties, grouped by the three routes described above plus the closest paired baseline (skill-creator). No outcome supervision: revision selection uses no external outcome signal; Pair: matched with/without-skill runs; Skill: revises a loadable skill package; Local edits: passage-level edits, not a wholesale rewrite.

Method	No outcome supervision	Pair	Skill	Local edits
Trajectory memory	✗	✗	✗	✗
Outcome-gated	✗	✗	✓	✗
Deployment-feedback	✗	✗	✓	✗
skill-creator	✗	✓	✓	✗
SELFEVOSKILL	✓	✓	✓	✓

C EVOLUTION LOOP ALGORITHM

Algorithm 1 SELFEVOSKILL Evolution Loop

Require: Task description T, workspace W, initial skill S_0
Ensure: Evolved skill S^*

$\mathcal{I} \leftarrow$ TASKINTERPRETER(T) {explicit constraints from the task}
$V \leftarrow$ COMPILETASKVERIFIER$(\mathcal{I}, W)$ {W grounds task-referenced data}
$(\tau_w, \tau_{wo}) \leftarrow$ PAIREDEXECUTION(T, W, S_0)
$D_0 \leftarrow$ PACE(τ_w, τ_{wo}, S_0) {first paired diagnosis}
mode $\leftarrow$ SELECTMODE(EFFECTIVENESSDELTA(D_0), DIAGNOSISSCOPE(D_0)) {lock Refine or Repair}
$S \leftarrow S_0$; $S_{\text{acc}} \leftarrow S_0$; $H \leftarrow \varnothing$
for $t = 1, \ldots, 5$ **do**
  if $t > 1$ **then**
    $\tau_w, \tau_{wo} \leftarrow$ PAIREDEXECUTION(T, W, S)
    $D_t \leftarrow$ PACE(τ_w, τ_{wo}, S)
  else
    $D_t \leftarrow D_0$
  end if
  verdict $\leftarrow$ AGGREGATE(D_t) {hurt has veto; mode unchanged}
  $H \leftarrow H \,\|\, [\textit{verdict}]$ {append to verdict history}
  if $S \neq S_{\text{acc}}$ **and** (*verdict* = *hurt* **or** $V(\tau_w)$ fails) **then**
    $S \leftarrow S_{\text{acc}}$ {reject the edited candidate}
  else
    if $V(\tau_w)$ passes **then**
      $S_{\text{acc}} \leftarrow S$
    end if
    if $t < 5$ **and** HASACTIONABLEPASSAGES$(D_t, \textit{mode})$ **then**
      $S \leftarrow$ SKILLITERATOR$(S, \tau_w, \tau_{wo}, \textit{mode})$
    end if
  end if
  if $S = S_{\text{acc}}$ **and** CONVERGED$(D_t, V(\tau_w), H)$ **then**
    break
  end if
end for
return S_{acc}

AGGREGATE follows the PACE precedence rule in §3.4; the Task-Compiled Verifier remains a separate acceptance gate. CONVERGED requires the verifier to pass, no *hurt* verdict in the last two iterations, and no actionable passages.

The pair computed before the loop is the first paired audit of S_0 and is reused in iteration $t = 1$. Its Effectiveness Delta and diagnosis scope select the locked mode; later iterations never reroute Refine and Repair. A *hurt* verdict or verifier failure rejects only a candidate produced by SKILLITERATOR

Table 7: Which updates Refine and Repair permit. Both use paired trajectories and the Task-Compiled Verifier.

Dimension	Refine	Repair
Current workflow	Kept	May be replaced
Default action	Delete, clarify, or correct locally	Replace the prescribed workflow
Protected passages	Marked by PACE to keep; not edited	All passages are evaluated by PACE; diagnosed protected passages are retained
Harmful content	Reject the update	Locate via PACE, replace or delete
Missing information	Local details only (unit, header)	Added after harmful passages are removed

and restores S_{acc}. Otherwise, actionable passages identified by PACE trigger an edit under the locked mode, including when the first audit selects Repair for an initially harmful or inert skill. The edited package is evaluated by the with-skill execution of the next iteration. The loop terminates early when no actionable passages remain, recent updates are not harmful, and the verifier passes.

D PACE Evaluator Inventory

PACE is built from a fixed inventory of twelve evaluator templates spanning the four dimensions introduced in §3.4. Table 8 gives one representative evaluator per dimension, and Table 9 gives the complete inventory. The same evaluator inventory is used for every paired audit; evaluators without relevant evidence return no signal or mark the criterion as not applicable. Each evaluator reads both τ_w and τ_{wo}, compares tool calls, decisions, and artifacts, and emits passage-level `action_signals` and `protected_hints` as described in §3.4.

Evaluator instantiation and information boundary. PACE combines this fixed inventory with task-specific grounding. Before evolution, the evaluator generator binds every immutable template to the explicit requirements in T, the declared workspace data in W, and the passage structure of the current skill. During each paired audit, every evaluator jointly examines τ_w and τ_{wo}, traces observed behavioral differences to specific skill passages, and emits structured `action_signals` and `protected_hints`. A protocol-level validator enforces outcome-supervision-free evolution by rejecting any evaluator payload containing task rewards, benchmark tests, benchmark-verifier outputs, reference solutions, or historical comparison reports. Paper-formatted versions of `eval-output-evidence-check` and `eval-skill-incremental-value` appear below; the supplementary artifact provides all twelve templates and their binding procedure.

Table 8: PACE evaluation dimensions, with one representative evaluator per dimension. Each evaluator compares τ_w and τ_{wo} along one dimension and outputs passage-level diagnostic signals.

Dimension	Representative evaluator	What is assessed
Process Adherence	eval-skill-procedure-adherence	Does the agent follow the skill's prescribed steps and tool-use patterns?
Artifact Evidence	eval-output-evidence-check	Do the output files meet explicit task requirements and deterministic workspace checks?
Consistency	eval-skill-task-alignment	Does the skill's prescribed workflow match the task's data schema and output format?
Effectiveness Delta	eval-skill-incremental-value	Does τ_w improve on τ_{wo} in calls, decisions, or artifacts?

Table 9: Complete fixed PACE evaluator inventory (12 templates across four dimensions). Every template is applied to each paired audit; a task without relevant evidence yields no signal or a not-applicable record.

Dimension	Evaluator	What it checks
Process Adherence	eval-skill-procedure-adherence	Does the agent execute the skill's prescribed steps in order, respecting preconditions and prohibited actions?
	eval-skill-coverage	What fraction of the agent's key actions are guided by explicit skill instructions vs. self-inferred?
	eval-tool-use-rationality	Are tool choice, call timing, parameter construction, and use of returned values reasonable?
	eval-error-robustness	Does the agent handle errors and exceptions sensibly rather than failing silently or retrying without bound?
Artifact Evidence	eval-output-evidence-check	Three-way comparison of observable tokens across the with-skill artifact, the without-skill artifact, and workspace data, with skill attribution.
	eval-format-compliance	Does the output conform to the format contract (structure, field completeness, types, enum validity)?
Consistency	eval-skill-task-alignment	Do the skill's instructions contradict the task requirements (over-specification, workflow conflicts)?
	eval-skill-data-consistency	Do the skill's data assumptions (file/column/field names, parameter values) match the actual data?
	eval-method-adherence	Does the agent use the specific method, algorithm, or tool the task mandates?
	eval-safety-compliance	Does execution violate a hard constraint whose breach is not recoverable within the task (unsafe paths, credentials, data protection, or destruction of content the task requires to be preserved)? Highest priority; vetoes on a critical violation.
Effectiveness Delta	eval-skill-incremental-value	Does the with-skill run provide a genuine procedural gain over the without-skill run, and does the gain come from the skill rather than native model ability?
	eval-skill-portfolio-quality	Audits skill-portfolio health: redundancy, document bloat (line/file inflation), and unilateral growth of imperative instructions.

Complete representative evaluator prompt: `eval-output-evidence-check`. Among the twelve fixed evaluators, `eval-output-evidence-check` supplies the artifact-level evidence used by PACE. It performs a three-way comparison among declared workspace data, artifacts produced with and without the skill, and the skill passages associated with the observed behavior. When task-grounded artifact evidence resolves the comparison, this evaluator is decisive in PACE aggregation. The paper-formatted prompt below preserves the complete decision procedure and output contract; symbolic paths are replaced with task-specific values during binding.

Role and objective. You are the Artifact Evidence evaluator in PACE. Determine whether the artifacts produced by the with-skill and without-skill executions satisfy the file-level and content-level requirements grounded in the task and workspace. Compare both conditions, identify skill-induced gains or regressions, and connect every actionable finding to the responsible skill passage.

Bound inputs. The evaluator receives: (i) the task description T; (ii) task-grounded `requirements`, `observable_tokens`, expected output files, hard constraints, and a profile of declared workspace data W; (iii) the current skill S; (iv) the matched trajectories τ_w and τ_{wo}; (v) their respective artifact directories; and (vi) the destination for the structured report. Task binding replaces these fields without changing the checks or decision rules below.

Evidence policy. Use only observable facts from T, W, S, the paired trajectories, and their generated artifacts. Inspect files with controlled reads, exact search, and format-aware parsers. Every finding must include a reproducible file excerpt, parsed value, or trace record; never invent an event or artifact. Do not access benchmark tests, benchmark-verifier outputs, scoring scripts, reference solutions, task rewards, or historical comparison reports. Task requirements are grounded in T and W; S is used to attribute observed behavior to an editable passage.

Evaluation procedure.

1. **Artifact existence and completeness.** For every expected output, check that the file exists, is nonempty, and can be parsed when its required format is structured, such as JSON, CSV, or XLSX.
2. **Three-way observable-evidence comparison.** For every task-grounded observable token, locate its source value in W, test its presence or required transformation in each output, and locate the passage in S associated with that behavior. A source value present in W and correctly propagated to an artifact is a pass; a source value present but missing from the artifact is a failure; an absent source value makes the check not applicable.
3. **Numerical and structural consistency.** Check explicit cross-field relations stated in T, including sums, percentages, identifiers, schemas, and preservation requirements. When the declared task structure supports an additional consistency comparison, record it as warning-level evidence rather than a hard requirement.
4. **With-skill versus without-skill comparison.** Apply the same checks independently to both artifact sets. Evidence present only in the with-skill artifact indicates a skill-enabled gain. Evidence present in the without-skill artifact but missing or degraded in the with-skill artifact is a candidate `skill_introduced_regression`.
5. **Skill attribution.** For every failure or regression, match the affected requirement and trace behavior to the responsible skill file, section heading, and verbatim passage. The passage location must be specific enough for the Skill Iterator to edit without rewriting unrelated content.
6. **Regression localization.** For a high-confidence regression, connect the affected evidence to the most recent relevant skill change and report the changed passage, affected token or invariant, and observed mechanism. Prefer restoring the last supported passage when the evidence identifies a regression.

Required output. Emit one structured record with the following fields:

- `evaluator_name`, fixed to `eval-output-evidence-check`; `dimension`, fixed to `artifact_evidence`; and `agent_mode`, identifying the with-skill, without-skill, or paired summary record.
- `findings[]`, where each entry contains the requirement identifier, checked token or invariant, source presence, artifact presence, result, confidence, responsible skill passage, and concrete evidence.
- `regression_findings[]`, where each entry contains the regressed token or invariant, the `skill_introduced_regression` verdict, confidence, responsible skill passage, and the associated skill change.
- `completeness_score`, `token_pass_rate`, overall result, severity, attribution, and a passage-specific suggestion.
- `action_signals[]`. For each high-confidence skill-induced regression, emit `action=investigate_regression`, the exact skill file and passage, confidence, and an evidence-grounded rationale. Otherwise emit an empty list.
- `protected_hints[]`, emitted as an empty list by this evaluator; preservation signals are supplied by the evaluators responsible for confirmed procedural value.

Aggregation contract. When task-grounded artifact checks resolve the comparison, return the artifact verdict as decisive evidence to PACE. Every signal must remain traceable from a task requirement or declared workspace fact, through the paired artifacts, to the cited skill passage.

Complete representative evaluator prompt: `eval-skill-incremental-value`. This evaluator operationalizes the Effectiveness Delta dimension of PACE. It compares matched executions, distinguishes skill-grounded behavioral differences from native-model variation, and links each supported difference to the responsible skill passage. Its comparison record supplies the incremental-value judgment used by the initial routing rule. The paper-formatted prompt below preserves the decision procedure and output contract while omitting serialization details.

Role and objective. You are the Effectiveness Delta evaluator in PACE. Determine whether loading the current skill provides a genuine procedural contribution beyond the frozen agent's behavior without that skill. Compare the matched trajectories, identify the passages associated with supported behavioral differences, and separate skill-grounded effects from native-model variation.

Bound inputs. The evaluator receives: (i) the task description T and declared workspace data W; (ii) the current skill S and its passage structure; (iii) the matched trajectories τ_w and τ_{wo}; (iv) their generated artifacts; and (v) the destination for the structured records. Task binding may identify expected divergence points, but it does not change the comparison rules below.

Evidence policy. Use only observable calls, decisions, tool results, and artifacts from the matched executions. Do not access task rewards, benchmark tests, benchmark-verifier outputs, reference solutions, or historical comparison reports. Attribute a difference to the skill only when the observed behavior matches a concrete

operation stated in a specific skill passage. Otherwise record the source as native-model behavior or unclear; such differences do not authorize a skill edit.

Evaluation procedure.

1. **Map passages to behavioral signatures.** Read the skill and associate each procedural passage with observable signatures such as tool choice, call order, arguments, paths, or artifact transformations.
2. **Identify paired divergences.** Compare τ_w and τ_{wo} at corresponding task steps, using tool calls, tool results, decisions, and artifacts to locate where their behavior diverges.
3. **Determine the source.** Mark a divergence as `skill_knowledge` only when the with-skill behavior matches a cited skill passage and the without-skill run lacks the corresponding behavior. Mark an ungrounded difference as `model_native`, and use `unclear` when the available trace cannot distinguish the source.
4. **Determine incremental value.** Return `positive` when a skill-grounded divergence improves an observable task step or artifact; `negative` when it introduces an error, detour, or degraded artifact; `zero` when both runs behave equivalently; and `zero_missing_knowledge` when the skill omits needed guidance that the agent supplies from native capability. If the traces are incomplete, return `unclear` and do not force a conclusion.
5. **Inspect unresolved stalls.** For a zero or unclear comparison, locate the last productive tool call on each side and record whether either execution makes materially deeper progress before repeated errors, abandonment, or prolonged absence of productive tool use.
6. **Produce passage-level signals.** A positive, skill-grounded difference protects the cited passage. A negative, skill-grounded difference marks that passage for deletion or replacement and, when available, records the corresponding verbatim tool-use fragment from τ_{wo}. Zero, native-model, and unclear differences do not create an edit target.

Required output. Emit one record for each execution and one comparison record. Each record contains `evaluator_name`, `dimension`, `agent_mode`, a factual `phenomenon`, traceable `evidence`, and any identified error. The comparison record additionally contains `incremental_value`, `incremental_source`, optional stall-depth fields, and `without_verified_snippets[]` for a supported negative difference. Emit `action_signals[]` only for negative differences attributed to skill knowledge, and emit `protected_hints[]` only for positive differences attributed to skill knowledge. Every signal must identify the skill file and quote the responsible passage verbatim.

Aggregation contract. PACE accepts exactly one comparison record from this evaluator. A positive comparison supplies positive Effectiveness Delta evidence, a negative comparison supplies negative evidence, and `zero` or `zero_missing_knowledge` supplies near-zero evidence. The initial routing rule combines this judgment with diagnosis scope to select Refine or Repair. An unclear comparison does not authorize an edit.

E Supplementary Evidence and Case Studies

Each case below illustrates one component of the method, stated in bold. Red marks the defective side of a contrast: a harmful instruction, or an observation that fails a constraint; blue marks the correct side: the without-skill reference, a corrected fragment, or an observation that satisfies the constraint. Cases whose claim depends on the loop include an iteration table. Figure 5 shows the constraint-check rollback and the later Repair steps on `data-to-d3`. Figure 6 compares the two pairing arms on `grid-dispatch-operator`. Cases A and B are the examples in §4.4. We will release the evolved skill packages with the implementation.

E.1 Case A: `software-dependency-audit` (Refine, primary)

Mode. Refine keeps a working workflow and deletes passages that τ_w does not use.

Offline Trivy scan of `package-lock.json` into a CSV with a fixed header. Opus evolved reward 1.0.

Contrast. Hardcoded `./trivy-cache` vs. `--cache-dir` with a parameter. Protected segments keep `--skip-db-update` and `--offline-scan`. Without-skill omits those flags. The prescribed scan workflow is not replaced.

Table 10: Four Opus case studies. Each row illustrates one mechanism: A, Refine keeps a working workflow; B, the framework rejects a candidate whose with-skill execution violates an explicit constraint; C, an inert round does not switch the fixed mode; D, a hurt verdict vetoes the update without recomputing the mode.

Task	Mode	Lines	Δ (%)	Paired contrast	Opus task reward
`software-dependency-audit`	Refine	935→610	−35	iter 2: τ_w PASS / τ_{wo} FAIL	1.0
`data-to-d3`	Repair	188→50	−73	iter 1 revert; iter 3: τ_w PASS / τ_{wo} FAIL	1.0
`lab-unit-harmonization`	Refine	408→259	−37	iter 2: τ_w PASS 8/8 / τ_{wo} FAIL 7/8	1.0
`exceltable-in-ppt`	Repair	1825→1773	−3	iter 2: τ_w FAIL (formula hurt) → rollback; iter 4: converge	1.0

Iter	τ_w	τ_{wo}	Decision	Lines
1	PASS	PASS	delete unused tutorial; replace cache path; keep offline flags	935 → 610
2	PASS	FAIL	*skill_helped*; stop	610

E.2 Case B: `data-to-d3` (Repair, primary)

Mode. The framework rejects a candidate whose with-skill execution violates a constraint already satisfied without the skill, and restores the previously accepted skill.

Build `/root/output/index.html` with D3 bubbles and a 50-row table whose column strings must match the task verbatim. The compiled verifier instantiates 13 checks, including the verbatim column-string requirement. At iteration 1, Title Case headers violate that check and yield a 12/13 result; the corrected artifact later passes all 13 checks. Opus 0.0 → 1.0. Figure 5 is the recorded process; the table lists every archived round.

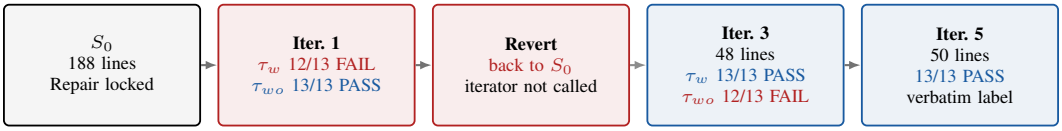


Figure 5: `data-to-d3`: the framework rejects a candidate whose with-skill execution fails a constraint already met without the skill, then continues Repair under the same mode. Dashed: iteration 2 is not in the archived excerpt.

Iter	τ_w	τ_{wo}	Decision	Lines
1	12/13 FAIL (columns)	13/13 PASS	revert to S_0; no iterator	188
2	—	—	not in archived excerpt	—
3	13/13 PASS	12/13 FAIL	*skill_helped*	48
4	12/13 FAIL (Title Case)	unstable	journal: agent styling, not skill	48
5	13/13 PASS	—	verbatim-label reminder; converge	50

Contrast. With-skill Title Case headers (`Ticker`, `Company Name`) vs. without-skill verbatim columns (`Ticker symbol`, `Full company name`). Iteration 1 rolls back on that contrast. Later Repair keeps tooltip and click snippets and adds a verbatim-label reminder.

E.3 Case C: `lab-unit-harmonization` (Refine)

Mode. An inert round does not switch the locked Refine mode; the paired contrast can appear later.

SI unit harmonization of a clinical CSV. Opus evolved reward 1.0. The first paired audit judges the with-skill run to provide positive incremental value, so the locked mode is Refine; the *skill_inert* verdict in iteration 1 records that no actionable surgery target was found in that round, not that the initial edit-mode choice changes.

Iter	τ_w	τ_{wo}	Decision	Lines
1	PASS 8/8	PASS 8/8	*skill_inert*; no edit; mode stays Refine	408
2	PASS 8/8	FAIL 7/8	protect unit table; delete unused text; *skill_helped*	259

Contrast. Iteration 1 has no failure that requires Repair. Iteration 2: τ_w converts a legacy `mg/dL` field vs. τ_{wo} omits that mapping. The unit-mapping table stays; surrounding unused passages are deleted.

E.4 Case D: `exceltable-in-ppt` (Repair)

Mode. A hurt verdict vetoes the current update and does not recompute the locked mode. This case is not evidence for the Task-Compiled Verifier.

Embed an Excel sheet into PowerPoint while keeping six cell formulas. Curated and evolved rewards are both 1.0. Here the first paired audit finds no positive incremental value, so the locked mode is Repair, even though both runs of iteration 1 pass the verifier as in Case C.

Iter	τ_w	τ_{wo}	Decision	Lines
1	PASS 6/6	PASS 6/6	*skill_inert*; no edit	1825
2	FAIL 0/6	PASS 6/6	*skill_hurt*; rollback; mode stays Repair	1825
3	PASS 6/6	PASS 6/6	delete `recalc.py` step; *skill_helped*	1773
4	PASS 6/6	PASS 6/6	verification; converge	1773

Contrast. `MANDATORY IF USING FORMULAS: Use the recalc.py script` zeroes formulas on this task (0/6). The hurt veto rolls back and leaves the mode Repair. A later Repair iteration deletes that step (−52 lines). The formula check that fires is *eval-safety-compliance*, not the Task-Compiled Verifier.

E.5 Case E: `paper-anonymizer` (Repair; passage-level diagnosis)

Mode. PACE names the passage to edit. Repair; the task's reward rises from 0 to 1 after evolution (no-skill 0, initial skill 0, evolved skill 1; Table 13).

Anonymize three academic PDFs for blind review (`/root/paper{1-3}.pdf` → `/root/redacted/`): redact author names, affiliations, arXiv identifiers, and accepted venues while preserving the References section. The task specification requires that at least 60% (0.6) of the source text be retained.

Contrast. Initial skill (`academic-pdf-redaction/SKILL.md`). Its retention guard is both self-contradictory and stricter than the task, in prose and in code:

```
VERIFY output - Check that 80%+ of original text remains
if redc_len < orig_len * 0.7:  raise ValueError(...)
```

The prose (80%) and the code (0.7) disagree, and both exceed the task's 0.6 floor, so the guard rejects a faithful redaction as a failure. PACE marks the guard as edit targets ST-002/ST-003 and separately unwraps a generic `redact_with_pymupdf(...)` function wrapper (ST-001), protecting the remaining sections. After the edit the guard reads:

```
Check that 60%+ of original text remains
if redc_len < orig_len * 0.6:  raise ValueError(...)
```

A later round confirms that the diagnosed issues have been resolved. During the initial audit, the diagnosis authorizes removal of off-domain files bundled with the skill (1223 → 112 lines); the in-domain redaction module is not rewritten.

E.6 Case F: `grid-dispatch-operator` (Refine; pairing contrast)

Mode. **A without-skill run is required for the update signal.** Two with-skill runs can treat an execution accident as skill harm and edit a working workflow. GLM pairing run (§4.3), not the Opus 89-task table. Figure 6 places the two arms side by side.

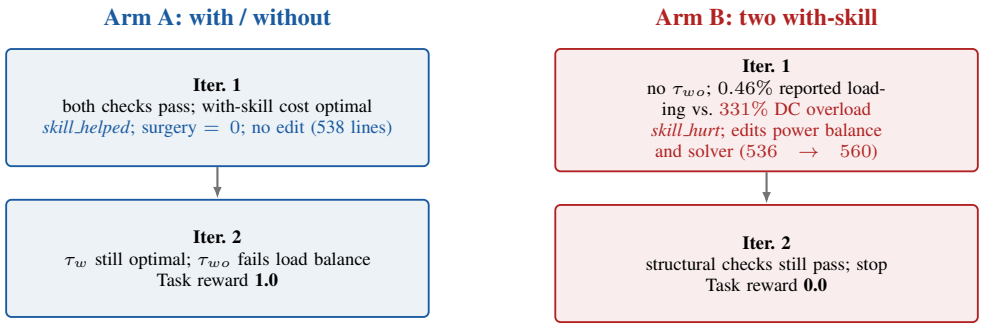


Figure 6: `grid-dispatch-operator` in the GLM pairing run. Arm A sees that unloading the skill is worse and leaves the working workflow unchanged. Arm B has no without-skill reference, treats a silent infeasibility as skill harm, and rewrites passages that still pass structural checks.

Contrast. Arm A: with-skill cost \$2,965,637.91/hr (optimal) vs. without-skill \$3,086,631.85; unloading the skill is worse, so the package is not edited. Arm B: a 0.46% reported loading against a 331% DC overload is treated as skill harm. The solver rewrite still passes structural checks; task reward is 0.0 (cost error ≈ 4%; tolerance 0.01%).

E.7 Structural Analysis Details

Domain-level reward profile. Figure 7 complements the task-level analysis in §4.4 by displaying the mean reward profile behind Table 1. The evolved skill leads the curated skill in seven of eight domains; the variation motivates examining which forms of skill knowledge leave diagnostic traces in trajectories and artifacts.

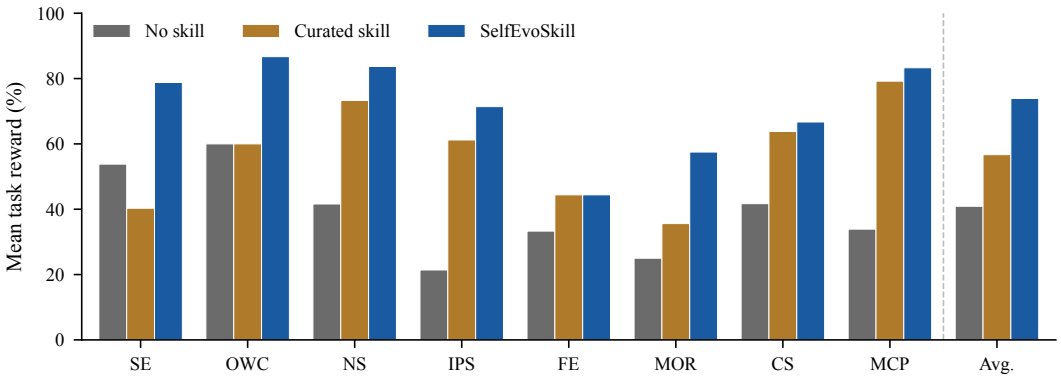
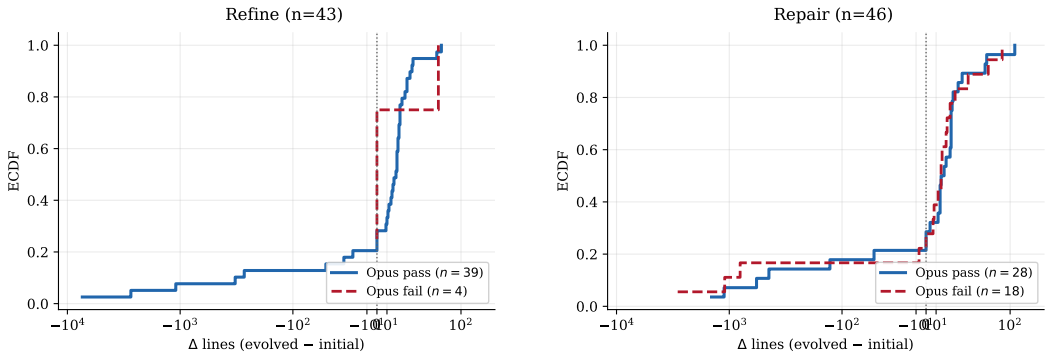


Figure 7: The evolved skill leads the curated skill in seven of eight domains. Mean task reward (%) by domain from Table 1. Domain codes match Appendix F. The dashed line separates the eight domain means from the overall mean across all 89 tasks.

Table 11 summarizes the line-count dynamics by edit scope: Refine has a median of +19 lines on passing tasks alongside a deletion tail reaching −7,375, while Repair makes more targeted swaps

Table 11: Skill line-count change (Δ = evolved − initial, all `.md` files) by edit scope. Refine's long deletion tail reflects that it may not replace the prescribed workflow; Repair makes more targeted edits. Throughout this appendix, *pass* denotes an evolved task reward ≥ 0.5. Each entry pairs task reward with the line-count change of the same single evolved skill: the run that completed without an infrastructure abort. We do not use the best-of-two reward from §4.1, because it does not correspond to a single skill. Figure 8 uses the same convention.

Edit scope	N	Pass / Fail	Median Δ (pass)	Mean Δ (all)	Min Δ
Refine	43	39 / 4	+19	−260	−7,375
Repair	46	28 / 18	+16	−165	−2,867



(a) Refine ($n = 43$): pass $n = 39$, fail $n = 4$.

(b) Repair ($n = 46$): pass $n = 28$, fail $n = 18$.

Figure 8: Empirical CDFs of skill line-count change (Δ = evolved−initial) for all 89 tasks, stratified by edit scope and Opus pass/fail. Solid: pass; dashed: fail. Dotted line marks $\Delta = 0$; symmetric log scale (linear for $|\Delta| \leq 50$).

with fewer extreme deletions. Figure 8 gives the full per-task distribution of skill line-count change, as empirical CDFs stratified by edit scope and post-evolution Opus pass/fail. The horizontal axis uses a symmetric log scale so that large deletions (e.g., `flink-query`, −7,375 lines) remain visible without compressing the central mass near zero.

F Task Manifest, Per-Task Results, and Label Statistics

Per-label statistics. Table 12 reports the average evolved reward for every knowledge-type and task-type label. Labels are not mutually exclusive: per-label counts are over the tasks bearing each label and do not partition the 89 tasks.

Among knowledge types, file format knowledge (82.4%) and data cleaning procedure (85.7%) attain the highest evolved rewards, while evaluation protocol (66.7%, $n = 3$) and tool workflow (68.4%) are the lowest. Among task types, generation (100%, $n = 5$) and transformation (88.9%) lead, whereas search (25.0%, $n = 4$) and repair (45.0%, $n = 8$) lag behind. These per-label patterns are consistent with the main-text finding (§4.4) that skills with observable execution signals benefit most from paired revision.

Frozen task manifest and per-task results. Table 13 lists the task identifiers that define the fixed evaluation snapshot and reports the per-task rewards behind Table 1 across the following pages. Dashes mark missing runs (two no-skill, one with-skill); they count as 0 in all averages (§4.1). Non-integer rewards are shown to two decimal places. Domain codes: SE = Software Engineering, OWC = Office & White Collar, NS = Natural Science, IPS = Industrial & Physical Systems, FE = Finance & Economics, MOR = Mathematics & OR, CS = Cybersecurity, MCP = Media & Content Production.

Three patterns are visible before inspecting individual rows. First, Software Engineering shows the largest domain-level improvement: evolved reward rises to 78.8% from a

Table 12: Per-label average evolved reward (%). *Left*: knowledge-type labels. *Right*: task-type labels. Counts are over tasks bearing each label (multi-label).

Knowledge type	n	Reward
domain procedure	58	69.2
library api usage	33	79.2
mathematical method	28	80.7
tool workflow	19	68.4
file format knowledge	17	82.4
debugging heuristic	8	70.0
data cleaning procedure	7	85.7
evaluation protocol	3	66.7

Task type	n	Reward
analysis	24	69.0
calculation	20	72.8
transformation	18	88.9
detection	15	80.0
implementation	15	80.0
optimization	12	80.0
extraction	11	63.6
repair	8	45.0
planning	8	57.5
simulation	7	71.4
debugging	6	76.7
classification	6	83.3
generation	5	100.0
verification	5	80.0
search	4	25.0
formatting	3	100.0
ranking	1	0.0

curated-skill baseline of 40.3%, with six tasks flipping from 0 to 1. Second, Repair-mode tasks account for most reversals in which the evolved skill falls below the stronger baseline, including `crystallographic-wyckoff` (evolved 0.55 vs. curated 1) and `latex-formula-extraction` (evolved 0 vs. no-skill 1). Third, Finance & Economics contains the highest concentration of zero-reward tasks across all three conditions, consistent with the tied domain result in Table 1.

Table 13: Per-task rewards for all 89 tasks. No skill / Curated skill / Evolved are the rewards of the corresponding condition; dashes mark missing runs, counted as 0 in all averages. Non-integer rewards are shown to two decimal places.

Task	Dom.	Mode	No skill	Curated skill	Evolved
`azure-bgp-oscillation-route-leak`	SE	Repair	0	0	0
`data-to-d3`	SE	Repair	0	0	1
`debug-trl-grpo`	SE	Refine	0.60	0.60	0.60
`dialogue-parser`	SE	Refine	1	1	1
`fix-build-agentops`	SE	Refine	0	0	1
`fix-build-google-auto`	SE	Repair	–	0	0
`fix-visual-stability`	SE	Refine	1	1	1
`flink-query`	SE	Refine	1	0	1
`jax-computing-basics`	SE	Refine	1	0	1
`llm-prefix-cache-replay`	SE	Refine	0	1	1
`parallel-tfidf-search`	SE	Refine	1	1	1
`python-scala-translation`	SE	Repair	1	0	0
`react-performance-debugging`	SE	Refine	0	0	1
`simpo-code-reproduction`	SE	Refine	0	0	1
`spring-boot-jakarta-migration`	SE	Repair	1	1	1
`tictoc-unnecessary-abort-detection`	SE	Refine	1	0.85	1
`citation-check`	OWC	Refine	1	1	1
`court-form-filling`	OWC	Repair	1	0	1
`edit-pdf`	OWC	Repair	0	0	1
`enterprise-information-search`	OWC	Repair	0	0	0
`exceltable-in-ppt`	OWC	Repair	1	1	1
`jpg-ocr-stat`	OWC	Refine	1	1	1
`latex-formula-extraction`	OWC	Repair	1	0	0
`offer-letter-generator`	OWC	Refine	1	1	1
`organize-messy-files`	OWC	Refine	1	1	1

Continued on next page

Table 13 *(continued)*

Task	Dom.	Mode	No skill	Curated skill	Evolved
paper-anonymizer	OWC	Repair	0	0	1
pdf-excel-diff	OWC	Repair	1	1	1
powerlifting-coef-calc	OWC	Refine	1	1	1
pptx-reference-formatting	OWC	Repair	0	1	1
sales-pivot-analysis	OWC	Repair	0	1	1
taxonomy-tree-merge	OWC	Refine	0	0	1
crystallographic-wyckoff-position-analysis	NS	Repair	0.55	1	0.55
earthquake-phase-association	NS	Refine	1	1	1
earthquake-plate-calculation	NS	Repair	0	1	1
exoplanet-detection-period	NS	Repair	1	1	1
find-topk-similiar-chemicals	NS	Repair	0	0	0
flood-risk-analysis	NS	Refine	0	1	1
glm-lake-mendota	NS	Repair	1	1	1
gravitational-wave-detection	NS	Repair	–	–	1
lab-unit-harmonization	NS	Refine	0.69	1	1
lake-warming-attribution	NS	Refine	0	1	1
mars-clouds-clustering	NS	Refine	1	1	1
protein-expression-analysis	NS	Refine	0	1	1
quantum-numerical-simulation	NS	Repair	0	0	0
radar-vital-signs	NS	Repair	0	1	1
seismic-phase-picking	NS	Refine	1	0	1
3d-scan-calc	IPS	Refine	1	1	1
ada-bathroom-plan-repair	IPS	Refine	0	1	0
adaptive-cruise-control	IPS	Repair	0	1	1
drone-planning-control	IPS	Refine	0	0.57	0
dynamic-object-aware-egomotion	IPS	Repair	0	0	0
energy-ac-optimal-power-flow	IPS	Refine	0	0	1
energy-market-pricing	IPS	Repair	1	1	1
energy-unit-commitment	IPS	Refine	0	1	0
grid-dispatch-operator	IPS	Refine	0	1	1
hvac-control	IPS	Refine	1	0	1
manufacturing-codebook-normalization	IPS	Repair	0	1	1
manufacturing-equipment-maintenance	IPS	Repair	0	1	1
manufacturing-fjsp-optimization	IPS	Refine	0	0	1
r2r-mpc-control	IPS	Repair	0	0	1
econ-detrending-correlation	FE	Refine	1	1	1
financial-modeling-qa	FE	Repair	0	0	0
invoice-fraud-detection	FE	Refine	1	1	1
reserves-at-risk-calc	FE	Repair	0	0	0
sec-financial-report	FE	Repair	0	0	0
shock-analysis-demand	FE	Repair	0	0	0
trend-anomaly-causal-inference	FE	Refine	1	1	1
weighted-gdp-calc	FE	Repair	0	1	1
xlsx-recover-data	FE	Repair	0	0	0
civ6-adjacency-optimizer	MOR	Refine	0	0.85	0.60
exam-block-sequencing	MOR	Refine	0	0	1
lean4-proof	MOR	Refine	1	1	1
paratransit-routing	MOR	Repair	0	1	1
pddl-airport-planning	MOR	Refine	0	0	0
pddl-tpp-planning	MOR	Repair	1	0	1
travel-planning	MOR	Repair	0	0	0
virtualhome-agent-planning	MOR	Repair	0	0	0
dapt-intrusion-detection	CS	Refine	0	1	1
fix-druid-loophole-cve	CS	Repair	1	0	0
fix-erlang-ssh-cve	CS	Repair	0	1	1
setup-fuzzing-py	CS	Repair	0.50	0.83	0
software-dependency-audit	CS	Refine	0	0	1
syzkaller-ppdev-syzlang	CS	Repair	1	1	1
mario-coin-counting	MCP	Refine	0	1	1
multilingual-video-dubbing	MCP	Repair	1	1	1

Continued on next page

Table 13 *(continued)*

Task	Dom.	Mode	No skill	Curated skill	Evolved
`pedestrian-traffic-counting`	MCP	Repair	0.03	0.75	1
`threejs-structure-parser`	MCP	Repair	0	1	1
`threejs-to-obj`	MCP	Refine	1	1	1
`video-silence-remover`	MCP	Repair	0	0	0