SKILLNEO: NON-ELITIST EVOLUTIONARY OPTIMIZATION OF AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills enable LLM agents to extend their capabilities without updating model weights. Skill evolution builds on this flexibility through continual skill refinement. Current methods use execution feedback to propose revisions, then select versions for further development based on current performance. However, a partially repaired skill may remain low-scoring until later edits resolve the remaining failures. To address this limitation, we introduce SkillNEO (Non-Elitist Evolutionary Optimization), which develops intermediate versions using comparative execution evidence. Its ordinary update selects parents from the current generation's evaluated children without requiring parent-relative gains or a place among the best historical versions. Comparisons of repeated, parent-child, and sibling executions identify unresolved failures and useful procedures. Meanwhile, archived versions can be revisited with later evidence, and their traces remain available for subsequent edits. We evaluate task-conditioned skill development on 86 SkillsBench tasks across eight domains, using GPT-5.4 Mini in every model-based development role. Selected bundles raise Mini's average reward from 49.1% to 66.4%, within GPT-5.5 and GPT-5.6 Luna's original-skill performance range. On the 41 modified tasks, frozen-skill reuse raises Luna's average reward by 10.5 percentage points; GPT-5.5 exceeds its original-skill reference by 10.6 points. Recorded searches show lower-scoring versions leading to later deployed bundles and supplying procedures to other branches. A weaker executor can thus develop skills reusable by stronger executors on the same tasks.

1 INTRODUCTION

LLM agents are increasingly used for diverse tasks, which often require domain-specific knowledge and procedures. General reasoning and tool access alone may not provide this expertise (Alzubi et al., 2026; Li et al., 2026). Agent skills enable LLM agents to perform specialized tasks by supplying procedural guidance (Wang et al., 2023; Li et al., 2026). Their forms include natural-language guidance, executable programs, workflows (Wang et al., 2025), and learned policies (Jiang et al., 2026). Skills support web operations (Zheng et al., 2025), shared interfaces across websites (Yu et al., 2026), memory management (Zhang et al., 2026c), and further skill development (Zhang et al., 2026a). Building on this, skill evolution refines skills over successive rounds (Zhang et al., 2026a; Alzubi et al., 2026; Zhou et al., 2026a). Each round revises skill contents using execution feedback and evaluates the revised versions (Tan et al., 2026; Gao et al., 2026). The resulting rewards guide selection for further revision. However, current reward may not reflect a version's development potential: completing a repair may require several interdependent edits. A revision can fix input processing while leaving output construction incomplete; its whole-task reward may remain unchanged or decline. Requiring improvement over the parent can exclude such a revision from further editing (Wang et al., 2026). Restricting future parents to the best historical versions can similarly exclude partially repaired alternatives (Alzubi et al., 2026). The question is how to continue developing these versions using their execution records.

Non-elitist evolutionary selection keeps the development population moving through new candidates (Beyer & Schwefel, 2002). In offspring-only replacement, unchanged parents leave the selection pool, so a high historical score does not itself preserve them. Candidates still compete on measured performance within their generation. This choice admits revision paths with temporary

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.

score decreases, but productive further edits also require evidence of what has changed. Prior work diagnoses skill behavior by comparing executions with and without a skill (Gao et al., 2026) and consolidates successful and failed trajectories into skill updates (Ni et al., 2026). We instead use the skill revision tree to compare executions across related versions. Parent-child comparisons relate behavioral changes to edits, while sibling comparisons contrast alternative repairs from a shared starting point. These comparisons can reveal working procedures within a failing bundle and unresolved steps that subsequent edits should address.

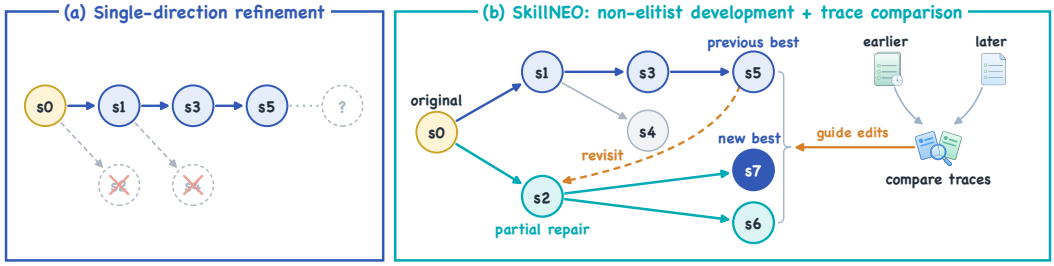


Figure 1: **Intermediate skill versions remain available for development.** States s_i pair skill bundles with their execution records. The dark-blue node marks the new deployment incumbent; crossed-out nodes in (a) denote discarded revisions. Solid tree edges denote edits; in (b), the dashed orange arrow denotes reselection.

We bring these ideas together in **SkillNEO**, a framework for *Non-Elitist Evolutionary Optimization* of complete agent skill bundles (Figure 1). Its ordinary parent update selects eligible children from the current generation for further development, without requiring parent-relative improvement or historical top-k membership. Comparisons of saved executions guide coordinated edits to instructions, scripts, and resources, including reuse of procedures from other branches. Archive revisiting extends development to earlier versions using evidence gathered after their creation.

We evaluate whether a weaker model's own executions can improve skills for itself and stronger executors. Prior work develops transferable skills with stronger-model guidance (Zheng et al., 2025; Zhang et al., 2026b; Yang et al., 2026); gskill develops transferable skills with a smaller executor and a stronger reflective proposer (Tan et al., 2026). DIVE reports cross-model reuse without a stronger teacher on mathematical and logical reasoning tasks (Xiong et al., 2026). In our study, GPT-5.4 Mini (OpenAI, 2026b) performs task execution, proposal generation, and editing. Across 86 SkillsBench tasks, its average reward rises from 49.1% to 66.4%, within GPT-5.5 and GPT-5.6 Luna's original-skill performance range. On the 41 modified tasks, frozen-bundle reuse improves Luna by 10.5 percentage points; GPT-5.5 exceeds its task-aligned published reference by 10.6 points.

Contributions. ❶ **We introduce SkillNEO for non-elitist skill evolution.** The framework combines selection among each generation's eligible children, archive revisiting, and comparisons across saved executions to guide coordinated revisions of complete skill bundles. ❷ **We show that weaker-model skill development can benefit stronger executors.** With Mini in every model-based development role, evolved bundles improve Mini and remain useful to GPT-5.5 and GPT-5.6 Luna on the same tasks. ❸ **We find that current reward can miss a revision's development value.** Recorded searches show lower-scoring versions leading to later deployed descendants, while execution comparisons identify unfinished repairs and procedures worth reusing.

2 Related Work

Search and candidate selection. Evolutionary comma selection replaces parents with offspring, allowing the active population's best fitness to decline across generations (Beyer & Schwefel, 2002). Prompt optimizers use critiques and beam search (Pryzant et al., 2023), error-guided tree search (Wang et al., 2024), prompt-score histories (Yang et al., 2024), or evolving mutation prompts (Fernando et al., 2023); NPO reports competitive single-lineage search in its evaluated settings (Chang & Chen, 2026). Skill-evolution methods differ in candidate admission and retention. Once its validation-ranked frontier is full, EvoSkill admits only candidates that outperform its lowest-scoring member (Alzubi et al., 2026). GraphSkillEvo uses mutation and crossover on graph-

structured skills, retaining the highest-scoring candidates from the combined parent-child pool (Sun et al., 2026). MetaSkill-Evolve retains failed revisions as inspiration but requires positive parent-relative gain for future parent eligibility (Wang et al., 2026). BONSAI revisits weaker branches through an evolvability-guided tree policy, supports grafting, and separates search from deployment; ordinary rewrites must improve on their parent on the reflection minibatch (Shastri et al., 2026). DIVE retains parents alongside children, targets lower-performing skills for exploratory revision, evolves independent populations, and selects complementary skills for inference (Xiong et al., 2026). DGM evolves coding agents by sampling archived implementations for further self-modification (Zhang et al., 2026d). In SkillNEO, ordinary updates select parents only from the current generation's eligible, fully measured children, without competition from previous parents or required parent-relative gains. Eligible archived bundles can return through separate revisiting.

Execution-guided skill refinement. AWM abstracts workflows from execution histories (Wang et al., 2025), while SkillRL co-evolves an experience-derived skill library with a reinforcement-learned policy (Xia et al., 2026). OpenClaw-Skill searches skill compositions across subtasks (Lin et al., 2026b); our tree records alternative versions of a complete bundle. Reflexion stores reflections for later attempts (Shinn et al., 2023), while ExpeL compares successful and failed executions to update reusable insights (Zhao et al., 2024). GEPA reflects on trajectories and merges prompts across lineages (Agrawal et al., 2026); EvoSkill and MetaSkill-Evolve reuse proposal histories or retrieved inspirations (Alzubi et al., 2026; Wang et al., 2026). Trace2Skill consolidates trajectory-level patches into a portable skill directory (Ni et al., 2026); SkillAudit compares executions with and without a skill to guide revision (Gao et al., 2026). Branch2Skill compares shared-prefix reasoning paths for local supervision (Ren et al., 2026); SkillLift learns an oracle-aligned rubric to guide revisions between costly evaluations (Kang & Wen, 2026). SkillNEO links executions to saved bundle versions, comparing repeated runs of the same version and parent-child or sibling executions. These comparisons guide coordinated edits to instructions and supporting files, including reuse of working procedures from revisions with unresolved failures. Revisiting archived bundles also lets later evidence inform new edits.

3 Problem Formulation

Let t denote a task with a fixed instruction, execution environment, and verifier V_t. SkillsBench supplies each task with a set of relevant skills (Li et al., 2026). Each skill contains its own `SKILL.md` document and optional executable tools and resources. A bundle L stores a version of this entire set, and SkillNEO evolves each task's bundle independently. A frozen agent A_m with model m executes t using L, producing a trajectory τ and an outcome y containing the submitted artifacts and final environment state. The verifier assigns reward $V_t(y) \in [0, 1]$. We define $J_m(L; t)$ as this agent's expected reward on task t with bundle L.

$$(\tau, y) \sim A_m(t, L), \qquad J_m(L; t) = \mathbb{E}_{(\tau,y)\sim A_m(t,L)}[V_t(y)]. \tag{1}$$

The expectation is over execution variability for the same task, model, and bundle. An execution record $e = (\tau, y, r, d)$ contains the trajectory, outcome, reward $r = V_t(y)$, and filtered verifier findings d. Traces and findings guide revisions; rewards measure task performance. Controlled bundle comparisons use the same task instruction, execution environment, verifier, and model configuration.

Starting from the original bundle L_t^0, edits define a space of reachable bundles $\mathcal{L}_t$. An edit $a = (\text{target}, \text{operator}, \text{specification})$ produces $L' = \text{Apply}(L, a)$ and may revise instructions, executable code, or resources within the bundle. The ideal objective and the output of a finite search are

$$\max_{L\in\mathcal{L}_t} J_m(L; t), \qquad \widehat{L}_t = \mathcal{S}_\kappa(t, L_t^0, m), \tag{2}$$

where $\mathcal{S}_\kappa$ denotes the search procedure and κ specifies its generation limit, candidate width, and evaluation repeats. We estimate $J_m(L; t)$ from a finite number of executions, retaining evaluated bundles and their execution records to guide subsequent edits. Once development ends for task t, we freeze its selected deployment bundle $\widehat{L}_t$. Running this procedure independently across tasks $\mathcal{D}$ yields the task-indexed assignment $\widehat{\mathbf{L}} = (\widehat{L}_t)_{t\in\mathcal{D}}$. Cross-model reuse then evaluates these same task-bundle pairs with another frozen model, without repeating skill development for the receiver.

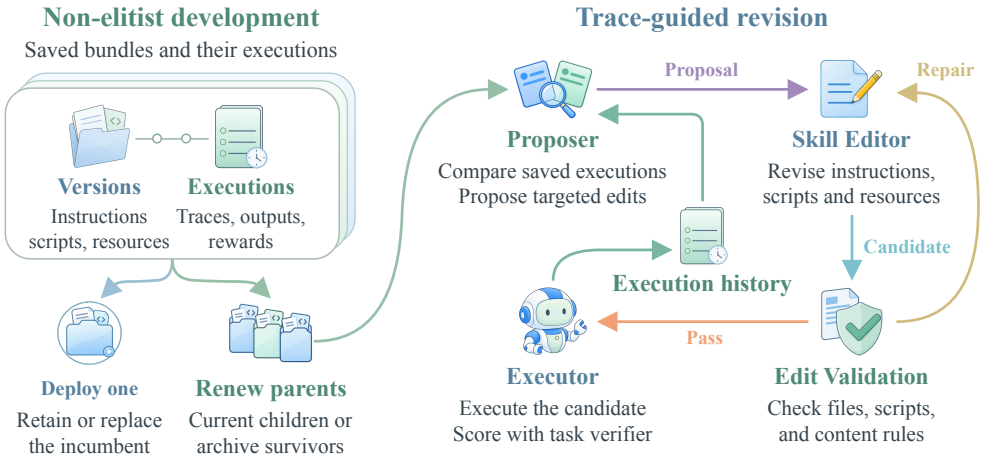


Figure 2: **Non-elitist skill development with comparative execution evidence.** Eligible parents (green) supply revisions; ordinary updates select from the current generation's evaluated children. Saved executions guide edits and remain available across revisits. Deployment selection (blue) uses a separate rule.

4 SKILLNEO

Figure 2 links non-elitist parent replacement to revisions guided by execution evidence. Each saved version pairs a complete bundle with its executions.

4.1 NON-ELITIST SKILL DEVELOPMENT

For task t, a rooted tree $\mathcal{T}_t = (\mathcal{V}_t, \mathcal{A}_t)$ records bundle revisions. Each node stores $s_v = (L_v, E_v)$, pairing a complete bundle L_v with execution records E_v from that snapshot. An edit a_u creates child u from a saved primary parent $p(u)$,

$$L_u = \mathrm{Apply}(L_{p(u)}, a_u), \qquad (p(u), u) \in \mathcal{A}_t. \tag{3}$$

The root stores $L_0 \equiv L_t^0$. A graft reuses a donor's procedure but retains one primary parent edge. At the end of generation g, h_g indexes the deployment incumbent, and B_g contains versions eligible to parent edits in generation $g+1$.

With parent-pool capacity k, up to k screened children are promoted for repeated evaluation (Section 4.3). Let C_g contain those that complete repeated evaluation and whose bundles remain available. The ordinary parent update is

$$B_g^{\mathrm{ord}} = \mathrm{Top}_k(C_g). \tag{4}$$

Previous parents do not participate in this pool: this is offspring-only comma selection (Beyer & Schwefel, 2002). Children compete on measured task reward within their generation without needing to outperform their parents or enter the historical top-k. Thus, children scoring below earlier bundles can remain eligible for further edits.

The resulting B_g^{ord} is the default parent set. Archive revisiting can replace it with eligible earlier versions under the rule in Section 4.3, returning development to earlier branches. The measured root is added if absent and fewer than two parents remain. These steps yield B_g for the next round of edits.

4.2 COMPARATIVE EVIDENCE FOR SKILL REVISION

Keeping a candidate editable leaves its next change to be determined from execution evidence. Generation one compares repeated root executions; later generations add population scores and evidence linked to bundle and parent versions: traces, outputs, artifacts, routing records, and edit

differences. Repeated executions reveal within-bundle variation, parent-child comparisons relate behavior to edits, and siblings contrast repairs from a shared starting point. The evidence pool includes screening-only children outside B_{g-1}, allowing earlier bundles to be revisited with later evidence.

The *Proposer* locates unresolved failures and successful procedures worth preserving. Routing records distinguish failure to consult a skill from failure after using it, guiding changes to its visible description or its procedure and supporting files. The prompt asks it to rank suspected causes by failed runs explained and check them against passing executions; recovered errors alone do not justify a revision. Each proposal identifies a primary parent from B_{g-1}, the intended change, and evidence, with verifier findings filtered beforehand. The prompt discourages near-repeats of edit directions measured as net losses, matching operator and target skill. Parent eligibility is determined separately by Section 4.1.

The *Skill Editor* creates a complete child bundle without modifying its saved parent. Edits can rewrite, add, remove, or reorganize instructions, scripts, and resources, or graft a procedure from another eligible parent. One objective may require coordinated file changes, such as adding a helper and the instruction invoking it. The editor is instructed to preserve unrelated content. Appendix B specifies operators and evidence inputs.

After editing, *Edit Validation* checks operation constraints, referenced files, new scripts, and candidate content. Fixable failures return to the Skill Editor for repair and rechecking; rejected candidates do not enter task evaluation. Admitted candidates receive benchmark verifier scores V_t. Appendix B.2 details the checks.

4.3 Evaluation and Search Schedule

The *Executor* produces trajectories, tool outputs, and artifacts; $\widehat{J}_v(Q)$ averages bundle L_v's verifier rewards over a nonempty batch Q (Equation 6). Each generation requests b children with per-parent cap $\lceil b/|B_{g-1}| \rceil + 2$. The root initially receives R executions, and each admitted child is screened once. With at least two children fully screened, up to k children are selected by screening reward for $R-1$ additional executions. The incumbent receives R fresh executions. Deployment ranks promoted children by their complete R-execution means, including screening. Compatible auxiliary scores break official-score ties at both stages. Our experiments use $(b, k, R) = (6, 3, 3)$.

Deployment compares the best fully measured child with the incumbent. The current-batch criterion requires an official-score gain beyond the margin, or an official-score tie with sufficient auxiliary gain. Under the strict-floor configuration, the child's mean must also exceed the larger pooled official score of the root and incumbent. Otherwise, the decision is a HOLD. Appendix B defines scores, margins, historical checks, and handling of incomplete records.

On a HOLD where the best child's official mean is below the incumbent's, enabled archive revisiting can replace the ordinary parent set with eligible saved versions. Retirement rules determine which versions can return; the incumbent remains deployed. Search allows up to five generations and stops after two consecutive holds. Appendix B gives the full archive, root-supplementation, and stopping rules; Algorithm 1 gives the complete procedure. The output $\widehat{L}_t = L_{h_g}$ is frozen for cross-model reuse.

5 Experiments

5.1 Experimental setup

Evaluation questions. We evaluate Mini's bundle improvement, frozen-bundle reuse by stronger executors, and development paths connecting intermediate versions to later revisions. Appendix A discusses the scope of these evaluations.

Tasks and aggregation. We evaluate on 86 professional tasks from SkillsBench 1.1, spanning eight broad categories (Li et al., 2026). Each task's supplied skills form its initial bundle, developed independently. The task set is $\mathcal{D}$. For model m and bundle assignment $\mathbf{L} = (L_t)_{t \in \mathcal{D}}$, repeat q produces $(\tau_{t,q}, y_{t,q}) \sim A_m(t, L_t)$.

We average three verifier rewards within each task and weight tasks equally.

$$S_m(\mathbf{L}) = \frac{100}{|\mathcal{D}|} \sum_{t \in \mathcal{D}} \frac{1}{3} \sum_{q=1}^{3} V_t(y_{t,q}). \tag{5}$$

Published references retain their documented aggregation. Average reward retains fractional rewards and is reported as a percentage; only binary tasks yield success rates.

Development and transfer. We use GPT-5.4 Mini/Codex (OpenAI, 2026a) at high reasoning for all LLM calls during development, including task execution, proposal generation, and skill editing. We do not use stronger models during development. Let $\mathbf{L}^0$ denote the original assignment and $\widehat{\mathbf{L}}$ the frozen assignment selected by development, containing 41 modified bundles and 45 bundles byte-identical to their originals. Cross-model reuse evaluates these same task-bundle pairs with GPT-5.4/Codex xhigh, GPT-5.5/Codex xhigh, and GPT-5.6 Luna/Codex max, without additional skill search or feedback to development.

Final evaluation. Mini's 41 modified bundles receive three post-evolution executions per task, separate from search and selection. The 45 unchanged bundles share the three original-skill executions between conditions; the 86-task score combines these with the modified-bundle measurements. Luna evaluates both assignments after skill freezing, yielding 516 valid executions. Appendix B gives the sources and execution details.

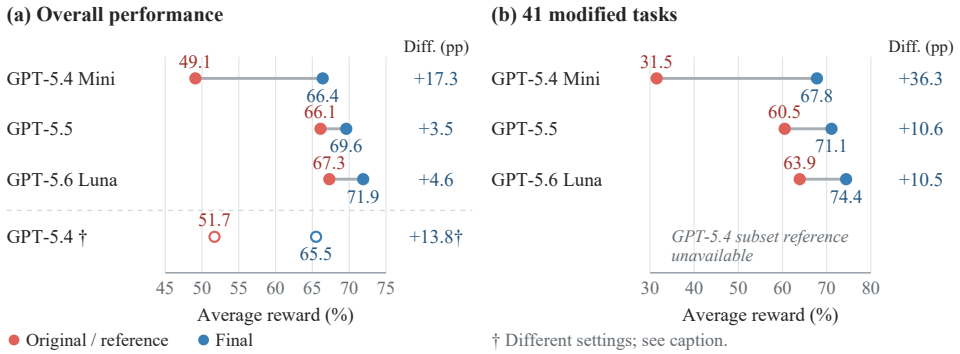


Figure 3: **Overall rewards and cross-model reuse.** Scores cover the 86-task assignment (a) and 41 modified tasks (b); differences are percentage points. † GPT-5.4 compares Vals (87 tasks, OpenHands) with our 86-task Codex evaluation; hollow, unconnected marks denote this setting mismatch. GPT-5.5 uses published, task-aligned SkillsBench references; Luna uses post-freeze evaluations of both conditions. Ties are within ±0.05 pp.

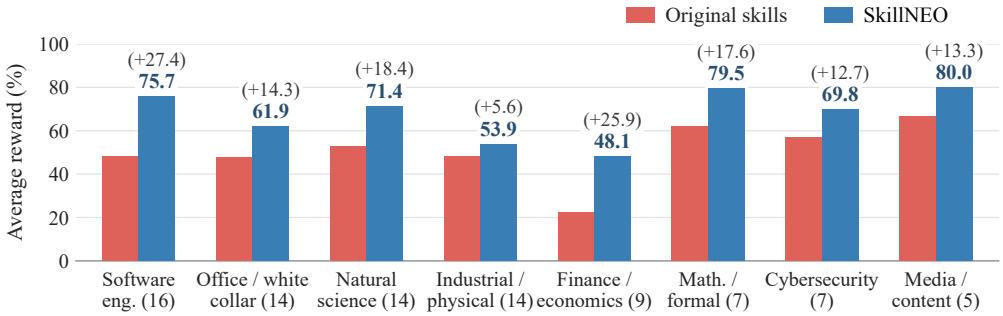


Figure 4: **Mini improves across all eight domains.** Original and final scores use Section 5.1's aggregation; gains are percentage points. Domain counts sum to 86.

5.2 Performance across tasks and domains

SkillNEO raises Mini's average reward across all eight domains; frozen bundles also improve Luna in post-freeze evaluation. Figure 3a summarizes overall results; panel (b) shows the 41-task modified subset. Scores report average reward (%); differences are percentage points (pp).

Mini reaches the frontier models' original-skill range. Mini scores 39.1 without skills and 49.1 with the supplied bundles (Table 2). Mini's original-skill score is 17.0 pp below GPT-5.5's published reference (66.1) and 18.2 pp below Luna (67.3). With Mini in every model-based development role, the final assignment reaches 66.4 (+17.3 pp), between those two original-skill scores. Fresh post-evolution evaluation of the 41 modified bundles gives 67.8 against the original-skill 31.5, with 29 gains, 12 ties, and no observed losses. The other 45 tasks retain their original bundles and Mini scores.

Execution records link skill revisions to workflow changes. Figure 4 shows gains in all eight domains, ranging from 5.6 pp in industrial / physical tasks to 27.4 pp in software engineering. Both domains begin near 48.3. Four cases connect revisions to execution changes and remaining obstacles. SimPO's original runs already produce loss vectors numerically matching passing runs, but use Python 3.12 despite the requirement for Python 3.10. The evolved skill puts environment activation before installation and testing; final executions satisfy this requirement and pass. The revised Data-to-D3 skill requires interaction checks. In one final execution, these checks reveal that selecting the same row twice clears its selection. The agent changes this behavior so that the row stays selected, then repeats the check. These revisions make an existing requirement part of the executed workflow or add a check that guides a repair before submission.

The other two cases reveal obstacles that remain after revision. Adaptive Cruise Control improves only partially. The remaining failed execution checks speed and distance summaries but does not verify that emergency mode occurs. Manufacturing Codebook Normalization's completed runs use test-item context where the revised skill requires component evidence from the raw segment for certain electrical-fault labels, and still fail codebook checks. In these tasks, final executions still leave a required behavior unchecked or a constraint unenforced. Appendix D.1 connects the skill changes to all 24 executions.

5.3 Cross-model reuse

Skills developed by Mini remain useful to stronger executors. On the 41 modified tasks, Mini-developed bundles raise Luna from 63.9 to 74.4 (+10.5 pp), with both conditions evaluated after freezing. GPT-5.5 scores 71.1 against its published original-skill reference of 60.5 (+10.6 pp). GPT-5.5 and Luna improve on six shared tasks and on 20 distinct tasks in total (Appendix C). These gains partly cover different tasks across receiving models.

Across all 86 tasks, Luna rises from 67.3 to 71.9. Modified bundles contribute +5.0 pp to this aggregate, while separately re-executed unchanged bundles contribute -0.4 pp (Appendix C). Luna's 41 modified tasks include 11 gains, 23 ties, and seven losses.

Workflow guidance can transfer despite Mini's remaining failures. A corrected workflow step can benefit receivers even when Mini fails elsewhere. Table 1 compares the requirements that four executors satisfy with identical frozen bundles. Mini stays below 100.0 on six of Luna's 11 improved tasks. In Software Dependency Audit, original-skill Mini and Luna scan beyond production dependencies; all final executions follow the revised scope. Mini rises from 0.0 to 66.7 despite a remaining output-field failure, while Luna rises from 0.0 to 100.0; GPT-5.4 and GPT-5.5 also reach 100.0.

Sales Pivot Analysis exposes complementary bottlenecks (Appendix F.1). Final Mini executions satisfy state-value checks but two omit Excel pivot objects; original-skill Luna builds pivots but includes an excluded category. The evolved skill specifies operation order and intermediate checks. Luna adapts the abbreviated whitelist example to the input's full state names and passes the required checks for state values and valid pivot objects.

Luna can follow the revised workflow through an adapted implementation. In one successful Sales Pivot execution, its workbook builder saves, reopens, and checks the source sheet before adding pivots. A later call to the bundled helper fails because a dependency is missing, yet the workbook already produced passes verification with full reward.

Table 1: Execution behavior with the same final skill bundle. Models differ in which requirements they satisfy despite receiving identical guidance. Counts give passes out of three executions; skipped checks are not passes. For Sales Pivot, *Valid pivot objects* requires the row-field, aggregation, and cache-field checks to pass. *Reward* is mean task reward (%).

(a) Software Dependency Audit

Model	Reward	Intended scan scope	Correct output fields
Mini	66.7	3/3	2/3
GPT-5.4	100.0	3/3	3/3
GPT-5.5	100.0	3/3	3/3
Luna	100.0	3/3	3/3

(b) Sales Pivot Analysis

Model	Reward	Valid state values	Valid pivot objects
Mini	33.3	3/3	1/3
GPT-5.4	66.7	2/3	3/3
GPT-5.5	0.0	0/3	3/3
Luna	100.0	3/3	3/3

5.4 Search Paths and Execution Evidence

The 41 modified bundles' histories document continued repairs, cross-branch procedure reuse, and revisions informed by later executions. Candidate letters `b` through `f` denote generations 1 through 5; `e04` is candidate 04 in generation 4.

Later deployed skills descend from lower-scoring candidates. Among the 29 tasks with final Mini gains, 22 (75.9%) use a non-incumbent parent on the final path; 16 paths pass through a candidate that scored strictly below the deployed bundle in the generation when it was created. At the candidate level, 163 such lower-scoring versions are expanded across the 41 searches; 45 have a later deployed descendant. For 18 of these 45 candidates, the first deployed descendant lies at least two edits away (Appendix H).

Gravitational Wave Detection shows how a further edit completes a partial repair in a lower-scoring bundle (Figure 5a). The task conditions detector data, runs matched filtering, and saves a CSV. Generation four deploys `e01` but retains zero-reward `e04` despite three higher-scoring saved versions. Each `e04` execution passes eight of nine checks but fails the numerical comparison. All three use 15 Hz for conditioning and search, although the search skill specifies 20 Hz. Its child `f04` consoli-

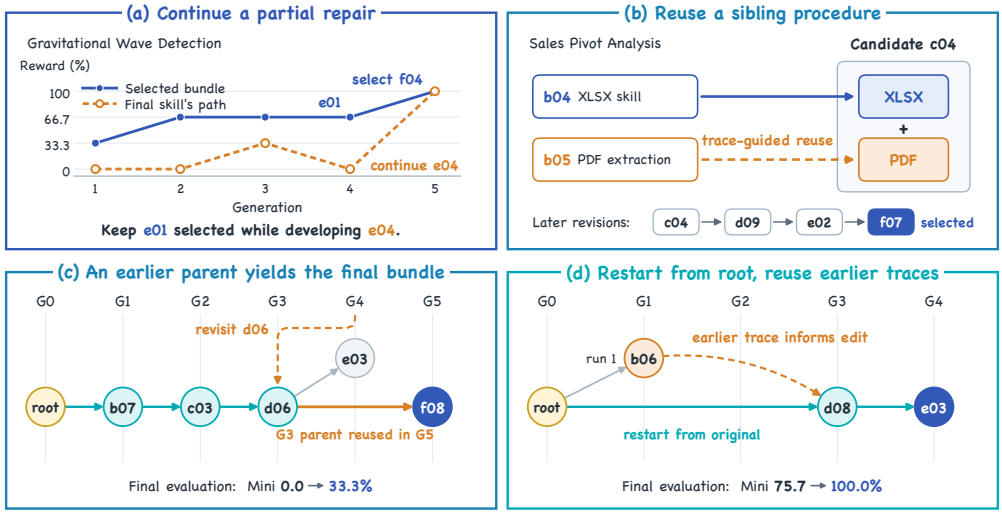


Figure 5: **Earlier revisions and executions support later development.** Panels (a) and (b) show three-run mean rewards at creation and sibling procedure reuse. Panels (c) and (d) show an earlier-parent revisit in Paratransit and a restart from root using later evidence in Wyckoff. In (c) and (d), solid edges show primary parents, broken arrows show reuse or reselection, and rewards come from independent final Mini evaluations (Appendices H and I).

dates the two skills, placing the 20 Hz search setting alongside the 15 Hz conditioning steps without changing helper code. All three child executions follow these settings and pass every check. The search deploys `f04` with 100.0% mean reward against the re-measured incumbent `e01`'s 66.7% (Appendix I.2). Final evaluation raises Mini from 0.0 to 100.0, while Luna retains 100.0. Data-to-D3 provides a second example of selection beyond the global top-k. Its candidate `d03` remains in a three-parent pool despite three higher-scoring saved versions, as detailed in Appendix H.

Whole-bundle performance can hide useful local procedures. Eight graft edits on six final ancestries explicitly draw from donors outside those primary paths. In Sales Pivot Analysis (Figure 5b), `b05` has lower three-run mean reward than `b04` despite useful state-name handling. One run succeeds; both failures pass state-validity checks but fail pivot configuration. The `c04` proposal cites the successful execution, selecting `b04` as parent and `b05` as donor. The edit retains `b04`'s spreadsheet skill for pivot construction and grafts `b05`'s PDF extraction procedure for state-name handling. The two passing `c04` executions use the adopted extraction pattern with full state names. The historical-score check retains `b04`, but `c04` remains a parent and enters the final ancestry. Appendix I.4 details the proposal, files, and executions.

Accumulated execution evidence guides later edits. Paratransit reselects `d06` (G3) after evaluating its G4 children; its G5 child `f08` becomes the final bundle, raising Mini from 0.0 to 33.3 (Figure 5c). The Wyckoff restart uses later evidence: in G3, the search returns to the original bundle and uses `b06`'s G1 execution to make file-based analysis and structured output requirements explicit in `d08`. A new execution reads and follows this guidance; `d08`'s G4 child `e03` becomes the final bundle, raising Mini from 75.7 to 100.0 (Figure 5d). Revisiting a saved version allows its next edit to use execution evidence gathered after that version was created (Appendix H).

Skill improvement can come from better use of existing guidance. Software Dependency Audit requires a vulnerability CSV restricted to production dependencies, as in its original command example. The `b06` revision adds deduplication and checks that the saved CSV matches the scan-derived keys. Its failed execution verifies 22 findings but still expands the scan to development dependencies. Passing executions report three production-dependency findings. Matching the CSV to the scan does not correct an overly broad scan scope. The next revision, `c01`, limits scan scope while retaining the reporting checks, addressing the upstream source of the incorrect records. All three search-time `c01` executions follow that scope and pass. Revisions in Software Dependency Audit and Gravitational Wave Detection make an existing choice explicit at its point of use, while the audit retains valid reporting checks.

Development changes how the 41 modified bundles divide guidance between instructions and executable support. `SKILL.md` files shrink by 35.0 lines per task on average, while support scripts grow by 74.0 lines; the bundles add 18 scripts. Total source length increases by 1.5% (Table 12). Erlang SSH illustrates simplification. A 485-line guide becomes a 31-line task procedure, and both Mini and Luna retain perfect final scores (Appendix I).

Remaining failures involve constraint enforcement, perception, and submission. The remaining failures in Section 5.2 concern constraint enforcement; receiver executions also expose perception and submission bottlenecks. In JPG OCR Statistics, Luna writes correctly structured workbooks but misreads receipt month tokens; revised parsing rules leave the perception error unresolved. In Paratransit Routing, two GPT-5.5 executions find and check better feasible solutions but fail to save them as the required output. Appendix F provides the receiver evidence.

6 Conclusion

SkillNEO develops agent skills through non-elitist parent selection and comparative execution evidence. Recorded searches show partial repairs leading to later deployed descendants and useful procedures moving across branches. Execution comparisons identify unresolved steps and procedures worth preserving. With GPT-5.4 Mini in every model-based development role, the evolved bundles improve Mini and remain useful to GPT-5.5 and Luna on the same tasks. These results support continued development beyond current reward. Future work should test reuse on unseen inputs and composition into shared skill libraries.

AI USE STATEMENT

In this work, generative AI tools assisted with manuscript polishing and parts of the code implementation, and supported the authors' analysis of experimental results.

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, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

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.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Nat. Comput.*, 1(1):3–52, March 2002.

Yuan Chang and Xiaoqi Chen. Naive prompt optimization: Rethinking the need for complex prompt search, 2026. URL https://arxiv.org/abs/2608.27266.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023. URL https://arxiv.org/abs/2309.16797.

Haowen Gao, Haoran Chen, Can Wang, Shasha Guo, Liang Pang, Zhaoyang Liu, Huawei Shen, and Xueqi Cheng. SkillAudit: Ground-truth-free skill evolution via paired trajectory auditing, 2026. URL https://arxiv.org/abs/2606.14239.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. SoK: Agentic skills – beyond tool use in LLM agents, 2026. URL https://arxiv.org/abs/2602.20867.

Haoxiang Kang and Ming Wen. Skilllift: Learning dense rubrics from sparse oracles for efficient skill evolution, 2026. URL https://arxiv.org/abs/2609.15396.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. SkillsBench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. MUSE-Autoskill: Self-evolving agents via skill creation, memory, management, and evaluation, 2026a. URL https://arxiv.org/abs/2605.27366v2.

Tianyi Lin, Chuanyu Sun, Jingyi Zhang, Changxu Wei, Huanjin Yao, Shunyu Liu, Xikun Zhang, Liu Liu, and Jiaxing Huang. OpenClaw-Skill: Collective skill tree search for agentic large language models, 2026b. URL https://arxiv.org/abs/2606.16774.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

OpenAI. Codex CLI, 2026a. URL https://learn.chatgpt.com/docs/codex/cli. Software documentation. Accessed September 17, 2026.

OpenAI. GPT-5.4 Mini model, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.4-mini. API model documentation. Accessed September 12, 2026.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494.

Yanwei Ren, Haotian Zhang, Likang Xiao, Jiaxing Huang, Jiayan Qiu, Baosheng Yu, Quan Chen, and Liu Liu. Branch2skill: Efficient skill evolution through reasoning trees, 2026. URL https://arxiv.org/abs/2608.08677.

Yash Priya Shastri, Anand Eswaran, Adnan Qidwai, Pankaj Thorat, and Sachin Joshi. BONSAI: Evolvability-guided tree search over skills, 2026. URL https://arxiv.org/abs/2608.07056.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Rui Sun, Zhi Zheng, Zhenkun Wang, and Zhichao Lu. Graphskillevo: Evolutionary optimization of graph-structured agent skills, 2026. URL https://arxiv.org/abs/2609.21749.

Shangyin Tan, Lakshya A Agrawal, Rohit Sandadi, Dan Klein, Koushik Sen, Alexandros G. Dimakis, and Matei Zaharia. Automatically learning skills for coding agents. GEPA Blog, Feb 2026. URL https://gepa-ai.github.io/gepa/blog/2026/02/18/automatically-learning-skills-for-coding-agents/.

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, 2023. URL https://arxiv.org/abs/2305.16291.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=22pyNMuIoa.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. MetaSkill-Evolve: Recursive self-improvement of LLM agents via two-timescale meta-skill evolution, 2026. URL https://arxiv.org/abs/2607.05297.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent Workflow Memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

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. In *ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving*, 2026. URL https://openreview.net/forum?id=FYc2IygegR.

Siheng Xiong, Ali Payani, Oguzhan Gungordu, and Faramarz Fekri. Dive: Unlocking self-improvement in frozen language models through diversity-driven skill evolution, 2026. URL https://arxiv.org/abs/2608.12486.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

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

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. PolySkill: Learning generalizable skills through polymorphic abstraction for continual learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=KdEsujyiSV.

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 (Steve) 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.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026d. URL https://openreview.net/forum?id=pUpzQZTvGY.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

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.

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

Yang Zhou, Zihan Dong, Zhenting Wang, Can Jin, Shiyu Zhao, Bangwei Guo, Difei Gu, Linjun Zhang, Mu Zhou, and Dimitris N. Metaxas. Evidence over plans: Online trajectory verification for skill distillation, 2026b. URL https://arxiv.org/abs/2605.09192.

A Limitations and Future Directions

Understanding skill search. Our experiments evaluate the complete framework, and recorded histories characterize how its development paths are used. Matched-budget comparisons are needed to isolate the effects of non-elitist selection, comparative execution evidence, and cross-branch reuse. Relevant selection policies include single-path refinement, selection from the union of current parents and children, and a persistent historical top-k frontier with the same number of parent slots. Disabling archive revisitation would assess its contribution within each policy. Such comparisons should use the same tasks, initial bundles, developer, verifier, model-call budgets, and task-execution budgets. Search currently stops after repeated deployment holds; future work could assess stopping criteria that track progress across the development population.

Generalization beyond development tasks. We evaluate frozen bundles on the same tasks and input instances used during development. Fresh executions measure performance on those instances

but do not establish generalization to unseen tasks or inputs. Future work should test whether these procedures remain effective on new task variants.

Search and developer capability. Mini performs all model-based development roles in this study. Extending the framework to developers with different capabilities and model families would help establish how search depth and developer capability jointly shape skill improvement. A central question is whether and when deeper search can compensate for limited developer capability, and which capabilities remain necessary for productive skill development.

Developing shared skill libraries. SkillNEO evolves the bundle supplied for each task. A shared library would need to accumulate improvements across tasks while managing interactions with previously learned procedures. Future work could study how to combine broadly useful procedures with task-specific guidance and assess revisions across the tasks that depend on them.

B Development Protocol

Development starts from the skills supplied for each SkillsBench task (Li et al., 2026), keeping their `SKILL.md` documents and optional tools and resources together as a bundle. Harbor executes tasks; SkillNEO compares traces, edits bundles, evaluates candidates, and selects versions.

The planned per-task configuration requests six children, promotes up to three, measures each promoted child and the incumbent over three repeats, and allows at most five generations. A full generation with six admitted children and three promotions contains six screening executions, six additional promoted-child executions, and three incumbent executions. Search width counts candidate bundles and is distinct from execution concurrency.

B.1 Search configuration

With generation limit G, width b, promotion count k, and repeat count R, the complete planned schedule contains $R+G[b+k(R-1)+R]$ task executions, giving 78 for $(G,b,k,R)=(5,6,3,3)$. Actual execution counts depend on the completed generations and measured candidates; proposal and skill-editing calls and final evaluations are separate. Algorithm 1 summarizes the loop.

The later-generation proposer chooses its primary parent from the eligible pool, with a per-parent cap of $\lceil b/|B_{g-1}|\rceil+2$. Graft donors are drawn from the same eligible pool, while execution evidence used for diagnostic comparisons also includes screening-only candidates. Diagnostic summaries include up to four bundle versions from each end of the ranking. The later-generation prompt lists edit directions previously measured as net losses and asks the Proposer to avoid closely similar directions with the same operator and target skill. This proposal-level constraint leaves parent eligibility to Equation 4 and the archive rules below.

Operators include rewriting, extending, pruning, removing, rerouting, adding executable support, and grafting from an eligible donor. Every child retains one primary parent edge, including grafted children. Output checks and deduplication can reduce the requested six children to fewer distinct bundles. Under configuration ϕ, the edit-admission check is $C_{t,\phi}(L_{p(v)}, L_v, a_v)$; its constraints and repair paths appear in Appendix B.2.

Execution feedback. The Proposer combines a fixed prompt template with agent-visible traces, rewards, and filtered verifier findings. Feedback filtering and candidate-content checks address evaluator-private answer material; public task requirements and observed behavior guide revisions. Appendix B.2 specifies these checks.

Screening and repeated evaluation. For bundle L_v, the mean reward over a nonempty execution batch Q with outcomes $y_{v,q}$ is

$$\widehat{J}_v(Q)=\frac{1}{|Q|}\sum_{q\in Q}V_t(y_{v,q}). \tag{6}$$

The execution batch Q identifies the runs used in each comparison. The root is measured on three repeats and becomes the first incumbent; search stops if all three receive perfect reward. Otherwise, every admitted child receives one screening execution. Infrastructure failures are separated from official timeouts, memory limits, and verifier failures. Incomplete records do not enter the ranking.

Algorithm 1 SkillNEO develops skill bundles through non-elitist replacement and comparative execution evidence.

Require: Task t, frozen actor A_m, original L_0; $b = 6$, $k = 3$, $R = 3$, $G = 5$
Require: Deployment configuration θ (including $\delta_{\text{off}}, \delta_{\text{aux}}$), validation configuration ϕ, archive flag
Evaluate root 0 on R repeats; initialize tree, $h_0 \leftarrow 0$, $B_0 \leftarrow \{0\}$, $z \leftarrow 0$, $g_{\text{stop}} \leftarrow 0$
if root is measured-perfect **then return** L_0
for $g = 1, \ldots, G$ **do**
    Assemble root evidence ($g = 1$) or previous-population and saved execution evidence
    Proposer specifies b edits from B_{g-1}; Skill Editor materializes candidate bundles
    Apply Edit Validation $C_{t,\phi}$ and permitted repairs; discard rejected drafts
    Save distinct admitted survivors D_g and their edit edges
    Screen D_g once; rank complete children by single-execution reward
    if fewer than two children have complete screening records **then**
        Interrupt with h_{g-1}, B_{g-1}, z unchanged; resume generation g after completion
    end if
    Promote up to k top-ranked screened children
    Add repeats $2{:}R$ to promoted children; measure h_{g-1} on R fresh repeats
    Check task, model, environment, and verifier consistency
    Rank complete R-execution means, including screening for promoted children
    if the incumbent has no complete comparison record **then**
        $h_g \leftarrow$ highest-ranked complete bundle
    else
        $c \leftarrow$ highest-ranked child; $h \leftarrow h_{g-1}$
        $d \leftarrow S_g(c) - S_g(h)$; $a \leftarrow A_g(c) - A_g(h)$
        $w \leftarrow [d > \delta_{\text{off}}] \vee [d = 0 \ \wedge \ a > \delta_{\text{aux}}]$
        $h_g \leftarrow c$ if w and $\text{Accept}_\theta(c, h; \mathbf{Q}_g, E_{\leq g}) = 1$; otherwise $h_g \leftarrow h$
    end if
    $g_{\text{stop}} \leftarrow g$
    $z \leftarrow 0$ if a new child is deployed; otherwise $z \leftarrow z + 1$
    $B_g \leftarrow B_g^{\text{ord}}$ from Equation 4
    On an enabled negative HOLD, use eligible archive survivors with R repeats first, then thinner records
    If archive revisit finds no survivor, retain the ordinary set or stop if it is also empty
    Append measured root if absent and $|B_g| < 2$
    if $z \geq 2$ or $(z \geq 1 \ \wedge \ 100 - S_g(h_g) < \delta_{\text{off}})$ **then break**
end for
return $L_{h_{g_{\text{stop}}}}$ ▷ Deployment incumbent

Screening ranks children by their single-execution official reward, using compatible auxiliary scores to break ties. Promotion requires at least two children with complete screening records. If fewer are available, execution of the generation is interrupted without a deployment decision or a hold-count update. The saved incumbent and parent set are retained, and the same generation can resume after the missing screening measurements are completed. Up to three top-ranked complete children retain their screening measurement and receive repeats two and three; their resulting three-execution means determine the deployment ranking. The incumbent receives three fresh repeats interleaved in the same comparison batch without using a promotion slot. Deployment compares only fully replicated records with matching task, model, environment, and verifier settings.

Deployment acceptance. For the best fully measured child c_g, deployment updates as

$$h_g = \begin{cases} c_g, & \mathrm{Accept}_\theta(c_g, h_{g-1}; \mathbf{Q}_g, E_{\le g}) = 1, \\ h_{g-1}, & \text{otherwise (HOLD)}, \end{cases} \tag{7}$$

where $\mathbf{Q}_g = (Q_{g,v})_v$ records the execution batch for each compared node and $E_{\le g}$ collects saved development evidence. The deployment configuration θ specifies the improvement margins and historical-score checks. These evidence arguments are suppressed below.

Write $S_g(v)$ for a bundle's official mean reward in the current comparison batch and $A_g(v)$ for its auxiliary score, both on a scale from 0 to 100. Candidates rank first by S_g, then by A_g. The auxiliary score uses the percentage of passed verifier tests only when the compared evaluations have compatible test coverage; otherwise it uses official reward. The incumbent wins an exact tie on both scores; candidate ties use a deterministic tie-break.

Let c be the highest-ranked fully measured child and h the incumbent. For the current decision, write $S(v) = S_g(v)$, $A(v) = A_g(v)$, $\Delta S = S(c) - S(h)$, and $\Delta A = A(c) - A(h)$. For the strict-floor rule, let $P_g(v)$ be the mean official reward on the 0–100 scale over saved executions of bundle v in this task search, up to and including the current comparison batch. This pool includes its child evaluations, subsequent incumbent re-evaluations, and root initialization where applicable. Records sharing an execution identifier are first averaged, so each distinct execution contributes once. At each deployment decision, recompute F_g as the maximum available $P_g(v)$ for $v \in \{0, h\}$; if neither pooled score is available, set $F_g = -\infty$. The strict-floor acceptance rule uses improvement margins δ_{off} and δ_{aux} for official and auxiliary scores and accepts a child according to

$$\mathrm{Accept}_\theta(c, h) = \mathbf{1}\big[S(c) > F_g \ \wedge\ \big(\Delta S > \delta_{\mathrm{off}} \ \vee\ (\Delta S = 0 \ \wedge\ \Delta A > \delta_{\mathrm{aux}})\big)\big]. \tag{8}$$

The improvement margins are configurable, with defaults $\delta_{\mathrm{off}} = 0$ and $\delta_{\mathrm{aux}} = 30$ pp. The auxiliary margin limits incumbent replacement due to small auxiliary-score fluctuations when official scores tie. Without a complete incumbent batch, selection uses the highest-ranked fully measured bundle.

Developmental selection and archive revisit. Ordinary parent selection takes the top k fully measured current-generation children whose complete bundles remain available, excluding the incumbent's repeat-evaluation entries. The previous parent pool does not enter this ranking, so its best member need not remain eligible. Neither parent-relative improvement nor acceptance for deployment is required. Rejected drafts and deduplicated bundles cannot become parents. If fewer than two parents remain after selection, the measured root is added if absent.

When enabled, archive revisit activates on a HOLD with $\Delta S < 0$. It retires fully measured children that score below the incumbent. It also retires a parent if that parent has an available score in the same comparison and all of its evaluated children score below that parent. Archive candidates are unretired nodes with no children remaining after deduplication and retirement. Eligible survivors with R repeats are ranked first by official score. If that tier is empty, the remaining measured survivors form the fallback tier. If no archive survivor exists, selection uses the ordinary parent set when available; otherwise search stops as exhausted. The measured root is then added if fewer than two parents remain and it is absent. This revisit occurs in nine decisions across seven tasks. Seven decisions use fully repeated survivors, and two use one-repeat fallback candidates.

Stopping. Search returns root if its initial measurements are perfect. Deployment updates reset the hold counter; holds increment it. Search stops after two consecutive holds, after a hold with $100 - S_g(h_g) < \delta_{\mathrm{off}}$, at the generation limit, or on archive exhaustion.

B.2 CANDIDATE CONSTRUCTION AND CHECKS

Edit Validation programmatically checks operation constraints, referenced files, new scripts, and introduced content. Fixable failures return to the Skill Editor; evaluation uses only admitted bundles.

The Proposer receives public instructions, agent-visible execution evidence, official rewards, and filtered evaluator findings. The Skill Editor receives the edit specification and current skill. Public task requirements and agent-computed results can guide task-specific changes; the checks target evaluator-private answers copied into deployed artifacts.

Before proposal generation, verifier-text filtering removes comparison-shaped answer text and known private literals. Redaction and rejection use reference material from test and specification files, excluding the reference solution. Where applicable, matching rules exclude public-instruction vocabulary and imported API names.

After editing, new `SKILL.md` lines and added or replaced support files are checked for exact private five-word sequences, normalized lines, literals, and distinctive identifiers. When a match triggers rejection, the Skill Editor receives a generic message without the matched answers. Fixable rejections permit one revision; a missing promised executable permits an additional repair limited to that file. Repaired artifacts are checked again, and rejected children do not enter evaluation.

A configurable content heuristic uses default rejection thresholds of 20 new bare numeric entries, or six new entries on lines containing values with at least four decimal places. Validation also checks for instructions that depend on unavailable grader messages and missing promised helper files. Script checks execute Python helpers and check syntax for other supported languages. Text-only revisions do not require a helper check. Broader grounding or skill-routing concerns may be reported without rejecting the candidate.

These checks target newly introduced content and specified matching patterns. Admitted bundles are then executed and scored by the task verifier.

Table 2: No-skill and original-skill references. Original skills increase average reward in each listed configuration on the aligned 86 tasks. Rewards are percentages and Δ is in percentage points.

Model / agent	No skill	Original skill	Δ
Mini / OpenHands (official)	29.9	40.7	+10.8
Mini / Codex high (this study)	39.1	49.1	+10.0
GPT-5.5 / OpenHands (official)	50.9	66.9	+16.0
GPT-5.5 / Codex (official)	46.2	66.1	+19.9

Official rows are reaggregated from the published per-task registry. Differences are computed from the displayed means. No Luna no-skill result has been measured.

B.3 EVALUATION SETTINGS AND COMPARISON CONTEXT

Final evaluation uses the frozen task-bundle assignment and the model settings in Section 5.1.

Reference source. The official rows in Table 2 and the GPT-5.5 original-skill reference use the published SkillsBench results, reaggregated over the 86 tasks. GPT-5.4's 51.7% original-skill reference comes from Vals and uses 87 tasks, OpenHands, and xhigh reasoning. The +13.8 pp difference between our 86-task Codex final score and this reference compares different settings.

Reported skill-development results on SkillsBench. Table 3 summarizes average reward reported by three prior studies alongside our results. Reported averages are 66.4% for SkillNEO, 53.42% for MUSE-Autoskill with GPT-5.5, and approximately 52% for Mini with SPARK skills, each under its own task set and evaluation protocol. SkillAudit reports 73.9%, using Opus 4.8 for task execution, PACE auditing, skill editing, and final evaluation (Gao et al., 2026).

Comparative auditing and continued development. The studies also differ in how they develop skills. MUSE-Autoskill creates skills from executions, and SPARK distills stronger-model trajectories. To repair supplied skills, SkillAudit (Gao et al., 2026) uses PACE to compare executions with and without the current skill, combines these diagnostics with a fixed structural verifier, and can roll back to the previous saved skill version. Its reported loop maintains one active skill state. SkillNEO

Table 3: Reported skill-development results on SkillsBench. Average rewards (%) use each study's task cohort and evaluation protocol. *Human* denotes each study's human-skill reference; *Developed* uses its resulting skills.

Study	Executor	Tasks	Human	Developed
SkillAudit (Gao et al., 2026)	Opus 4.8	89	56.7	73.9
MUSE-Autoskill (Lin et al., 2026a)	GPT-5.5	75	59.67	53.42
SPARK (Zhou et al., 2026b)	GPT-5.4 Mini	86*	≈ 47	≈ 52
SkillNEO (this study)	GPT-5.4 Mini	86	49.1	66.4

This table compares average reward. SkillEvolver and CoEvoSkills instead report complete-pass frequencies. SkillAudit, MUSE-Autoskill, and SkillNEO use the listed model for development and execution; SPARK uses stronger-model teacher trajectories; SPARK's scores follow Section 4.1 of the source. SkillAudit and SkillNEO refine supplied skills; MUSE-Autoskill and SPARK generate skills, with Human as a separate reference. MUSE counts its 28 uncovered tasks as zero. *SPARK describes an 86-task suite; exact Mini task alignment is unavailable. Task cohorts and protocols follow each source. For SkillNEO, 41 modified bundles receive new final executions; 45 retained bundles share their original Mini measurements.

also develops supplied bundles with Mini and permits continued editing of children without parent-relative improvement. Parent-child and sibling records organize evidence across revisions, and grafting allows procedures from another saved branch to enter a candidate. These mechanisms preserve alternatives both as starting points for further edits and as sources of useful procedures; Section 5.4 traces both uses in the recorded executions. SkillAudit excludes external task rewards during evolution, whereas SkillNEO uses benchmark rewards to evaluate candidates and select versions.

C Analysis Across Receiving Models

We analyze the three-repeat task means from the 86-task evaluation. The 41 modified tasks divide into 29 Mini-positive and 12 Mini-tied tasks under the ± 0.05 pp tie band; the other 45 tasks retain their original bundles. These Mini-outcome subgroups are defined by the final evaluation results.

Table 4: Receiver rewards grouped by Mini's outcome. GPT-5.5 and Luna have larger mean gains on tasks where Mini improves than on tasks where Mini ties. Scores report average reward (%); *Gap* is evolved minus original reward in percentage points.

		GPT-5.4	GPT-5.5			GPT-5.6 Luna		
Mini outcome	n	Evolved	Original	Evolved	Gap	Original	Evolved	Gap
All modified	41	66.8	60.5	71.1	+10.6	63.9	74.4	+10.5
Mini improved	29	69.1	64.3	78.1	+13.8	62.8	81.0	+18.3
Mini tied	12	61.1	51.4	54.2	+2.8	66.7	58.3	-8.3

The 41 modified tasks are grouped post hoc by Mini's final evaluation outcomes, using its ± 0.05 pp display band. Evolved denotes the same frozen Mini-developed bundle for each task. GPT-5.4 reports frozen-skill endpoints; original-skill subgroup scores are not reported. Evaluation settings follow Section 5.1.

Reward by Mini outcome. Table 4 reports receiving models' achieved rewards and, where available, differences from their original-skill references within these subsets. In the Mini-improved group, GPT-5.5 and Luna gain 13.8 and 18.3 pp, respectively. In the Mini-tied group, GPT-5.5 gains 2.8 pp while Luna loses 8.3 pp.

Why selected revisions can tie the original bundle in final evaluation. Selection compares a candidate with that generation's incumbent, whereas final evaluation compares the frozen bundle with the original skill in separate execution batches. Both the reference bundle and observed reward can therefore differ. Eleven of the 12 Mini-tied bundles were adopted with higher official reward than the incumbent; Quantum Numerical Simulation won an auxiliary-score tie-break while official reward remained zero. Grid was adopted at 100 with a 33.3-point incumbent gap, while its final original and evolved scores were both 66.7.

The final ties consist of four ceiling cases, three partial-score cases, and five zero-score cases. Perfect original reward leaves no room for observed improvement; when both conditions yield zero reward, neither resolves the recorded failures.

Absolute success and agreement. Among the 41 modified tasks, 14 achieve 100.0% average reward with all three receivers, 15 with one or two, and 12 with none. Achieving 100.0% requires all three saved repeats to receive full reward. Three tasks receive zero with all three receivers: Manufacturing Codebook Normalization, Paratransit Routing, and Quantum Numerical Simulation. Comparisons with original skills distinguish full reward from improvement for GPT-5.5 and Luna.

Table 5: Joint directions of receiver reward changes. Gains and losses need not coincide across GPT-5.5 and Luna on the 41 modified tasks. Rows use GPT-5.5 reference differences; columns use Luna differences from post-freeze evaluations.

GPT-5.5 direction	Luna gain	Luna tie	Luna loss	Total
Gain	6	6	3	15
Tie	4	13	1	18
Loss	1	4	3	8
Total	11	23	7	41

Both axes use the existing ± 0.05 pp display band. Original-skill reference sources appear in Appendix B.

Direction depends on the receiving model and its comparison. Table 5 groups tasks by the directions of their GPT-5.5 and Luna reward differences. Six tasks gain in both, while 20 gain in at least one. Four have opposite nonzero signs. Civ6 Adjacency Optimizer, Grid Dispatch, and Organize Messy Files decrease in both; Appendix F details each model's failures.

Table 6: Contributions to Luna's overall reward difference. Gains on modified tasks outweigh the decrease on retained tasks. Rewards are percentages; Δ and *Full-set contribution* are in percentage points. The Mini-positive and Mini-tied subsets partition the 41 modified tasks.

Stratum	n	Original	Evolved	Δ	Full-set contribution
Modified skills	41	63.9	74.4	+10.5	+5.0
Mini positive subset	29	62.8	81.0	+18.3	+6.2
Mini tied subset	12	66.7	58.3	-8.3	-1.2
Original retained	45	70.4	69.6	-0.7	-0.4

The 41 modified and 45 retained tasks partition the benchmark. Their contributions are $n\Delta/86$ and sum using unrounded values. Mini-positive/tied strata are defined post hoc from Mini's final evaluation outcomes.

Contributions to the full-cohort differences. Table 6 weights each subset's mean reward difference by its share of the 86 tasks. Luna's modified and retained subsets contribute approximately $+4.99$ and -0.39 pp to the $+4.60$ pp aggregate gain; the retained subset's own mean differs by -0.74 pp between separate original- and final-assignment execution batches, although its skill bundles are identical. GPT-5.5's contributions of $+5.05$ and -1.55 pp yield a $+3.50$ pp difference.

Table 7: Luna transfer by observed original performance. Among the 41 modified tasks, the group with zero original reward has the largest mean gain. Rewards are percentages, Δ is in percentage points, and *G/T/L* counts gains, ties, and losses.

Original-performance stratum	n	Original	Evolved	Δ	G/T/L
Zero observed reward	8	0.0	50.0	+50.0	4/4/0
Partial observed reward	16	57.5	61.4	+3.9	7/3/6
Perfect observed reward	17	100.0	98.0	-2.0	0/16/1

G/T/L counts gains, ties, and losses with the ± 0.05 pp display tolerance of Figure 3. Groups are defined post hoc from Luna's three original-skill repeats.

To relate reuse gains to starting performance, Table 7 groups the 41 modified tasks by their observed reward on Luna when using the original skills. Four of the eight tasks with zero original reward

improve and four tie. Outcomes are more mixed among the 16 tasks with partial original reward: seven improve, three tie, and six lose reward. Of the 17 tasks with perfect original reward, 16 retain perfect reward and one receives a lower reward.

Table 8: Mini's domain-level rewards. SkillNEO improves average reward across all eight domains. Scores are percentages for each domain's N tasks; Δ gives percentage-point differences before display rounding. The overall row reports the 86-task aggregate. Evaluation follows Section 5.1.

Domain	N	Original skills	SkillNEO	Δ
Software engineering	16	48.3	75.7	+27.4
Office / white collar	14	47.6	61.9	+14.3
Natural science	14	53.0	71.4	+18.4
Industrial / physical	14	48.3	53.9	+5.6
Finance / economics	9	22.2	48.1	+25.9
Mathematics / formal	7	61.9	79.5	+17.6
Cybersecurity	7	57.1	69.8	+12.7
Media / content	5	66.7	80.0	+13.3
Overall	86	49.1	66.4	+17.3

Table 9: Domain-level improvement and transfer. Mini improves in every domain, while Luna differences vary by domain. Differences are percentage points. *All* and *Modified* give task counts; the modified-task and full-domain differences use their respective denominators.

Domain	All	Modified	Mini mod.	Luna mod.	Luna all
Cybersecurity	7	4	+22.2	+44.4	+25.4
Finance / economics	9	4	+58.3	+25.0	+3.7
Industrial / physical	14	7	+11.3	-11.9	-10.7
Mathematics / formal	7	3	+41.1	-10.6	+0.2
Media / content	5	2	+33.3	+0.0	+13.3
Natural science	14	6	+42.9	+11.1	+9.5
Office	14	6	+33.3	+11.1	-2.4
Software engineering	16	9	+48.7	+14.8	+10.4

D Domain Results

Mini improves in every domain, while Luna's domain-level differences include gains and losses. Table 8 reports Mini's domain scores from Figure 4. Table 9 adds Mini and Luna differences on the modified-task subsets and Luna's full-domain differences.

D.1 Resolved and remaining workflow obstacles

To contrast resolved and remaining workflow obstacles, we examine original and final skills with three Mini executions per condition for four tasks (24 executions).

Removing an execution barrier. In SimPO Code Reproduction, original executions log Python 3.12, violating the required version 3.10 and stopping verification at the Python-version assertion. Comparing the saved outputs separately shows that all original runs' 100 loss values numerically match those from passing final executions. The evolved skill makes creating and activating the required environment precede installation, logging, and testing. All final executions use Python 3.10 and pass, raising reward from 0.0 to 100.0.

Data-to-D3 also rises from 0.0 to 100.0. Original executions write the required files and check syntax, but do not perform browser or DOM interaction checks; all fail tooltip behavior. The evolved skill explicitly requires checking the rendered page and linked interactions. All final executions perform interaction checks (two with Chromium and one with jsdom) and pass all 15 tests. One check reveals that repeated selection clears the selected row; the agent changes toggling to idempotent selection before submission. The skill guides the agent's construction of execution-time checks.

Failures can remain after a targeted edit. Adaptive Cruise Control rises from 33.3 to 66.7. Its evolved skill requires checking fresh simulation outputs, but names a helper path that is unavailable during execution. All final runs implement their own checks. The remaining failed run checks aggregate metrics but leaves emergency behavior absent, failing safety and scenario checks. The bundled checker omits these conditions and cannot verify the complete controller.

Manufacturing Codebook Normalization exposes inconsistent rule application, with a final reward of 0.0. The revised rule requires component evidence in the raw segment for certain electrical-fault labels. All final runs read the skill, but the two completed runs also admit contextual test-item evidence; one further replaces the required raw-text cue with a higher matching threshold. Both retain mismatched component assignments and pass 15 of 16 tests, failing the remaining codebook check based on lexical overlap. Another final run times out without the required output.

E Complete Task-Level Results

Table 10: Complete task-level rewards. Each row reports mean reward (%) on one of the 86 tasks. M denotes modified bundles; R denotes byte-identical retained bundles. O/E denote original/final assignments. Mini's R rows reuse the same original three-run score in both columns. Evaluation follows Section 5.1; reference sources appear in Appendix B.

Task		Mini O	Mini E	5.5 O	5.5 E	Luna O	Luna E
3d-scan-calc	R	100.0	100.0	100.0	100.0	100.0	100.0
ada-bathroom-plan-repair	R	33.3	33.3	100.0	100.0	100.0	100.0
adaptive-cruise-control	M	33.3	66.7	33.3	100.0	66.7	100.0
azure-bgp-oscillation-route-leak	R	0.0	0.0	0.0	0.0	0.0	0.0
citation-check	R	100.0	100.0	100.0	100.0	100.0	100.0
civ6-adjacency-optimizer	M	0.0	23.3	95.0	63.3	63.3	31.7
court-form-filling	M	33.3	66.7	33.3	66.7	33.3	100.0
crystallographic-wyckoff-position-analysis	M	75.7	100.0	100.0	100.0	100.0	100.0
dapt-intrusion-detection	R	100.0	100.0	100.0	100.0	100.0	100.0
data-to-d3	M	0.0	100.0	33.3	100.0	0.0	0.0
debug-trl-grpo	M	41.7	61.7	91.7	83.3	100.0	100.0
dialogue-parser	M	83.3	83.3	83.3	83.3	100.0	100.0
drone-planning-control	M	42.2	54.4	56.7	70.0	84.4	67.8
dynamic-object-aware-egomotion	R	0.0	0.0	33.3	0.0	66.7	0.0
earthquake-phase-association	R	66.7	66.7	100.0	100.0	100.0	100.0
earthquake-plate-calculation	R	100.0	100.0	100.0	100.0	100.0	100.0
econ-detrending-correlation	M	0.0	100.0	100.0	100.0	100.0	100.0
edit-pdf	M	0.0	33.3	100.0	100.0	100.0	100.0
energy-ac-optimal-power-flow	R	100.0	100.0	100.0	100.0	100.0	100.0
energy-market-pricing	M	66.7	100.0	33.3	66.7	66.7	0.0
energy-unit-commitment	R	0.0	0.0	100.0	0.0	0.0	0.0
enterprise-information-search	R	0.0	0.0	0.0	0.0	0.0	0.0
exam-block-sequencing	R	100.0	100.0	100.0	66.7	0.0	0.0
exceltable-in-ppt	R	100.0	100.0	100.0	100.0	100.0	100.0
exoplanet-detection-period	M	0.0	66.7	100.0	100.0	66.7	100.0
financial-modeling-qa	R	66.7	66.7	100.0	100.0	100.0	33.3
fix-build-agentops	M	66.7	100.0	66.7	66.7	100.0	100.0
fix-build-google-auto	R	0.0	0.0	0.0	66.7	66.7	33.3
fix-druid-loophole-cve	R	100.0	100.0	66.7	66.7	100.0	100.0
fix-erlang-ssh-cve	M	100.0	100.0	33.3	66.7	100.0	100.0
fix-visual-stability	R	100.0	100.0	33.3	0.0	66.7	100.0
flink-query	R	0.0	0.0	33.3	0.0	0.0	33.3
flood-risk-analysis	R	0.0	0.0	0.0	0.0	33.3	0.0
glm-lake-mendota	R	100.0	100.0	0.0	100.0	100.0	100.0
gravitational-wave-detection	M	0.0	100.0	100.0	100.0	100.0	100.0
grid-dispatch-operator	M	66.7	66.7	66.7	33.3	66.7	0.0
hvac-control	M	100.0	100.0	100.0	100.0	33.3	100.0
invoice-fraud-detection	M	0.0	100.0	0.0	100.0	0.0	100.0

Task		Mini O	Mini E	5.5 O	5.5 E	Luna O	Luna E
jax-computing-basics	R	100.0	100.0	100.0	100.0	100.0	100.0
jpg-ocr-stat	M	0.0	66.7	33.3	33.3	33.3	0.0
lab-unit-harmonization	R	100.0	100.0	100.0	100.0	100.0	100.0
lake-warming-attribution	M	66.7	100.0	66.7	33.3	66.7	100.0
latex-formula-extraction	R	33.3	33.3	0.0	0.0	100.0	0.0
lean4-proof	R	100.0	100.0	100.0	100.0	100.0	100.0
llm-prefix-cache-replay	R	100.0	100.0	100.0	100.0	100.0	100.0
manufacturing-codebook-normalization	M	0.0	0.0	0.0	0.0	0.0	0.0
manufacturing-equipment-maintenance	R	33.3	33.3	100.0	66.7	66.7	100.0
manufacturing-fjsp-optimization	R	0.0	0.0	33.3	0.0	100.0	66.7
mario-coin-counting	M	100.0	100.0	0.0	100.0	100.0	100.0
mars-clouds-clustering	M	33.3	66.7	100.0	100.0	100.0	100.0
multilingual-video-dubbing	M	33.3	100.0	33.3	100.0	100.0	100.0
offer-letter-generator	R	100.0	100.0	100.0	100.0	100.0	100.0
organize-messy-files	M	0.0	0.0	100.0	33.3	66.7	0.0
paper-anonymizer	M	0.0	33.3	100.0	100.0	100.0	100.0
parallel-tfidf-search	M	33.3	100.0	100.0	100.0	66.7	100.0
paratransit-routing	M	0.0	33.3	100.0	0.0	0.0	0.0
pddl-airport-planning	M	33.3	100.0	66.7	100.0	100.0	100.0
pddl-tpp-planning	R	100.0	100.0	100.0	100.0	33.3	66.7
pdf-excel-diff	R	100.0	100.0	100.0	100.0	100.0	100.0
powerlifting-coef-calc	R	100.0	100.0	100.0	100.0	100.0	100.0
pptx-reference-formatting	R	100.0	100.0	100.0	100.0	100.0	100.0
protein-expression-analysis	R	100.0	100.0	100.0	100.0	33.3	100.0
python-scala-translation	M	0.0	100.0	33.3	100.0	0.0	100.0
quantum-numerical-simulation	M	0.0	0.0	0.0	0.0	0.0	0.0
r2r-mpc-control	M	100.0	100.0	66.7	100.0	100.0	66.7
radar-vital-signs	R	66.7	66.7	100.0	100.0	66.7	100.0
react-performance-debugging	M	66.7	66.7	66.7	33.3	66.7	66.7
reserves-at-risk-calc	R	0.0	0.0	0.0	0.0	0.0	0.0
sales-pivot-analysis	M	0.0	33.3	0.0	0.0	0.0	100.0
sec-financial-report	M	33.3	66.7	66.7	0.0	66.7	66.7
seismic-phase-picking	R	33.3	33.3	100.0	100.0	100.0	100.0
setup-fuzzing-py	M	0.0	22.0	55.3	83.0	5.3	83.0
shock-analysis-demand	R	0.0	0.0	0.0	0.0	0.0	0.0
shock-analysis-supply	M	0.0	0.0	0.0	0.0	66.7	66.7
simpo-code-reproduction	M	0.0	100.0	0.0	100.0	100.0	100.0
software-dependency-audit	M	0.0	66.7	66.7	100.0	0.0	100.0
spring-boot-jakarta-migration	R	100.0	100.0	100.0	100.0	100.0	100.0
suricata-custom-exfil	M	0.0	0.0	100.0	100.0	100.0	100.0
syzkaller-ppdev-syzlang	R	100.0	100.0	100.0	100.0	100.0	100.0
threejs-structure-parser	R	100.0	100.0	100.0	100.0	33.3	100.0
threejs-to-obj	R	100.0	100.0	100.0	100.0	100.0	100.0
tictoc-unnecessary-abort-detection	M	81.7	100.0	100.0	100.0	100.0	100.0
travel-planning	R	100.0	100.0	100.0	100.0	100.0	100.0
video-silence-remover	R	0.0	0.0	0.0	0.0	0.0	0.0
weighted-gdp-calc	R	100.0	100.0	100.0	100.0	100.0	100.0
xlsx-recover-data	R	0.0	0.0	0.0	0.0	0.0	0.0

Table 10 reports the full 86-task comparison underlying the aggregate results. Each row corresponds to one task and its complete collection of skills. A modified marker denotes an evolved bundle; a retained marker denotes a byte-identical original bundle. Mini's 41 modified bundles use post-evolution final executions. Its 45 retained rows represent the same skill configuration in both columns and use the same original three-run score. Together these rows describe the final skill assignment for each task. Section 5.1 specifies aggregation; Appendix B gives the reference sources.

F CROSS-MODEL EXECUTION ANALYSIS

We first examine how evolved skills help receivers when the developer's final performance remains imperfect, then analyze unresolved failures. The matched Luna comparison yields 11 gains, 23 ties, and seven losses among the 41 modified tasks.

F.1 REUSABLE WORKFLOW IMPROVEMENTS AND RECEIVER EXECUTION

On six of Luna's 11 improved tasks, Mini's final mean reward is below 100.0. We examine Software Dependency Audit and Sales Pivot Analysis through 33 executions: 24 with final evolved skills, six with original skills on Luna, and three original-skill Mini audit executions. For each task, the final-skill executions use the same frozen bundle across all four models. Both tasks use binary execution rewards, so a mean of 33.3 or 66.7 denotes one or two successes out of three.

A shared workflow error is corrected, while a downstream failure remains. In the original-skill evaluations, all three Mini executions and two of Luna's three expand the scan to development dependencies and report 22 findings. Luna's remaining original execution keeps the intended scope but truncates a fix list. With the final audit bundle, `c01` (Appendix I.1), all 12 final executions across the four models keep the intended production scope and report three HIGH-severity findings. Mini rises from 0.0 to 66.7, and Luna from 0.0 to 100.0. Mini's failed final execution writes a CVSS score of 9.8 rather than the 8.1 required by the source-priority rule, and truncates a three-version fix list to one version. All three receivers preserve the required fields. The remaining Mini error concerns a pre-existing field rule, beyond the added row-count and record-key checks.

The receiver must apply the correction to the actual input. With the final Sales Pivot bundle, `f07` (development in Appendix I.4), all Mini executions pass state validation, but two write summary tables without actual pivot objects. Luna's original-skill executions create pivots but fail state validation by retaining `Other Territories`. With the evolved bundle, Luna filters to the eight full state and territory names in the input, excludes four out-of-scope records, and builds pivots over the remaining 2,450 regions. Its executions satisfy both requirements. This requires adapting the skill's whitelist example, which still uses abbreviations such as `NSW`. GPT-5.4's failed execution uses those abbreviations; its two successes retain full names. GPT-5.5 builds pivots but retains the excluded category in all three executions. These outcomes depend on adapting the guidance to the input's state representation and completing both requirements.

Reusable guidance can coordinate operations across saved artifacts. The final Sales Pivot skill also specifies dependencies between operations: clean missing income markers before computing quartiles, and write, save, reopen, and check the source sheet before attaching pivot objects. Mini's successful final execution computes quartiles over valid income values and maps the labels back to the merged data. Section 5.3 describes a successful Luna execution that follows the same sequence of saving, reopening, and checking through its own builder.

Table 11: Luna tasks with observed transfer losses. Seven modified tasks lose reward under evolved bundles. Values are mean rewards (%); *Change (pp)* uses unrounded measurements.

Task	Original	Evolved	Change (pp)
Energy Market Pricing	66.7	0.0	−66.7
Grid Dispatch Operator	66.7	0.0	−66.7
Organize Messy Files	66.7	0.0	−66.7
Civ6 Adjacency Optimizer	63.3	31.7	−31.7
Drone Planning Control	84.4	67.8	−16.7
JPG OCR Statistics	33.3	0.0	−33.3
R2R MPC Control	100.0	66.7	−33.3

F.2 UNRESOLVED FAILURES IN LUNA REUSE

For Luna's seven loss tasks, we compare three original-skill and three evolved-skill executions per task (42 total), together with the corresponding edits. Table 11 reports mean rewards.

Optimization and directory contracts. Energy and Grid each fall from two successes to none after independently replacing demand P_d with $P_d + G_s$, motivated by passing development evidence. All six evolved executions apply this rule and pass schema and feasibility checks, but fail cost optimality in the high-cost direction. The benchmark uses P_d. The four passing original executions use P_d; the remaining original execution of each task fails the cost check in the opposite direction. Passing development evidence therefore did not establish that the revised demand rule was correct.

Organize also falls from two successes to none. Every evolved execution places correctly classified documents in destinations below the source directory, `/root/papers/all/`. Its inventory, manifest, and exact-once audits check the destination tree, whereas the verifier treats documents remaining below the source directory as unsorted leftovers. The failed original execution makes the same choice, while the two passing ones use sibling destinations under `/root/papers/`. The stronger self-check leaves source-directory clearance unspecified. For Mini, Energy improves, while Grid and Organize tie their original-skill scores.

Saving the solution. Civ6's zero-reward executions (one original and two evolved) find competitive legal layouts but continue enumeration without saving the required JSON before timeout. Two other executions time out with valid outputs and receive 0.95 normalized adjacency reward. Missing artifacts account for the loss; the submitted layouts remain competitive. The evolved workflow lacks a requirement to save each new best layout before continuing. Its helper checks serialization and supplied-score sums but omits the stated adjacency recomputation. The observed missing-output failures occur before an artifact is available for that check.

Computation and perception. Drone's three low-scoring executions use quintic profiles; three high-scoring executions use clamped cubic interpolation. Although the task text permits broader continuous trajectories, the original skill and oracle specify cubic interpolation. The mismatch between these quintic trajectories and the oracle's cubic interpolation explains 13 common command failures, including all failures in the low-scoring original execution. The revised skills target hover and completion, but hover succeeds throughout, the cubic-planning skill is unchanged, and the revision does not resolve the choice between these planners. Scoring counts successful commands individually, so all enclosing pytest groups pass even at low reward.

JPG workbooks cover all 22 receipts with correct structure and monetary values. Five failed executions misread one or two month tokens on the same receipts across conditions; the errors do not involve swapping day and month. Replacing multi-pass guidance with deterministic parsing leaves perception errors despite image review; parsing cannot verify recognition accuracy.

The only failed R2R execution passes the structure, linearization, safety, and local artifact checks, but fails the verifier's settling-time test. The agent's local settling-time metric uses an all-channel absolute-error band and a post-step time origin; the verifier uses the changed roller's band relative to reference-step magnitude and the logged starting time. These metric conventions also differ in passing executions under both skill conditions; the task specifies a settling target without fully defining the measurement convention. Applying the verifier's rule to the failed execution's saved log reproduces its settling-time failure.

Saturation and unresolved tasks. Beyond these seven loss cases, the 23 tied tasks comprise 16 perfect, four zero, and three partial comparisons with two successes per condition. The zero ties are Data-to-D3, Paratransit Routing, Manufacturing Codebook Normalization, and Quantum Numerical Simulation; only the first two improved on Mini. The partial ties are React Performance Debugging, SEC Financial Report, and Shock Analysis Supply; only SEC improved on Mini. Perfect ties leave no room for observed gains; other ties include failures under both skill conditions.

F.3 Additional receiver evidence

Shared tasks, different failures. GPT-5.5 also loses reward on Grid, Organize, and Civ6. Grid has two failures at a 0.0111% cost gap against a 0.01% limit, yet every execution reads $P_d + G_s$, including the passing one. That instruction alone does not determine reward. Organize's failed executions misclassify one paper; only one also violates directory clearance, and another execution passes. Civ6 has one missing-output timeout and two 0.95 outcomes, showing the same checkpoint weakness with a different frequency.

Saving an improved solution. All three Paratransit GPT-5.5 executions time out, submitting reports serving 0, 440, and 442 trips against a minimum of 443. Two find and locally check better feasible reports under other filenames but fail to save them as the graded output. The remaining execution continues optimization after saving a below-threshold report. These executions expose the need to save the best available solution as the graded output.

Answer and execution checks. GRPO's two 0.75 GPT-5.5 outcomes fail only the epsilon source-pattern check. Their patches bind a named constant to 10^{-4}, whereas the verifier requires the denominator to contain `std_grouped_rewards` plus an inline numeric literal. The passing execution uses that literal. The quarter-point deduction is for the source-code pattern, even though the patch uses the requested numeric value.

Lake Warming's two GPT-5.5 failures attribute the dominant contribution to Human; the passing execution reports Heat. The attribution calculation differs across these outcomes, while formatting does not explain the difference. React's two failures pass ten other checks but violate the external-API check because checkout times of 8 and 11 ms fall below the 400 ms minimum. SEC's three normally completed executions fail answer checks on the newly acquired-position list in one case and on the number of reported holdings in two.

G Changes to Skill Artifacts

Table 12 summarizes changes between original and deployed skill bundles. Counts cover the files supplied to the agent, distinguishing core instructions from executable support. Each skill package is counted separately for each task that uses it. Across all 86 tasks, 62 of 228 task-skill pairs change (27.2%); within the 41 modified tasks, these 62 changes occur among 125 pairs (49.6%). Denominators use the union of original and evolved packages. Per-task means, medians, and correlations use the 41 modified tasks, whose total source length grows by 1.5%.

Table 12: Changes from original to deployed skill bundles. Core instructions become shorter on average while support scripts grow. The first two rows cover all 86 tasks; remaining statistics use the 41 modified tasks. Means and medians are per task.

Property	Value
Tasks with changed skill bundles	41/86 (47.7%)
Skill packages changed (counted per task)	62/228 (27.2%)
Total lines: original → deployed	106,632 → 108,233 (+1.5%)
Inserted + deleted lines: total / mean / median	8,147 / 198.7 / 157
Mean net line change per task: `SKILL.md`	-35.0
Mean net line change per task: support scripts	+74.0
Files added / deleted / modified	18 / 2 / 60
Line edits vs. reward change: ρ (Mini / Luna)	0.25 / 0.06

Packages and files are counted separately for each task; 228 counts the union of original and deployed packages. Line edits = insertions + deletions; net change = insertions − deletions. ρ is Spearman correlation. All added support files are scripts (17 Python, one shell).

Line edits count insertions plus deletions; net length change counts insertions minus deletions. Files are matched by skill and relative path within each task; a rename counts as deletion plus addition. Core instructions, scripts, and references are classified by their bundle paths.

H Search Histories and Development Paths

We trace each frozen non-root deployment back to its recorded search root along primary-parent links. The 41 task histories contain 1,283 candidate entries: 971 retained candidates and 312 duplicate entries removed by deduplication. Duplicate entries are not additional independently evaluated candidates. GPT-5.4 Mini performs diagnosis and writing throughout, including the 126 edits on final adopted paths. These search-process summaries cover the 41 modified tasks and exclude the 45 tasks that retain their original bundles.

Evaluation coverage. Decision rankings contain 705 means from three executions each, covering 529 candidate versions and 176 incumbent comparisons.

Actual expansion and later deployment. An expansion event identifies a task, generation, and parent with at least one observed child. It is non-incumbent when that parent differs from the deployment incumbent entering the generation. This label records parent identity within a run; the survivor-selection rule determines whether development is non-elitist. We count an event as leading to deployment when one of its generated children, or a descendant of that child, is selected for deployment in that generation or a later one. Eligibility alone does not count as an expansion event.

There are 418 expansion events, of which 289 use non-incumbent parents. Of these 289 events, 76 (26.3%) have a descendant later deployed, and 61 (21.1%) enter the final deployed ancestry; each outcome spans 30 tasks. Among the 76, 74 lead to a descendant accepted through a reward improvement or tie-break. The remaining two lead to Python-to-Scala deployments without a complete incumbent comparison. Deployment decisions are available for 178 of 180 groups defined by task and generation; unavailable decisions are not assigned outcomes.

Continuation from strictly lower-scoring candidates. We measure relative reward against the deployment selected in each candidate's birth generation. This comparison is distinct from a candidate's reward change relative to its own parent. Among the 529 fully measured candidate versions, 247 have a three-execution mean more than 0.05 percentage points below the deployment selected in the same generation's decision ranking. Both scores use that ranking's complete three-execution records, including the freshly measured incumbent when it remains deployed. Of these lower-scoring candidates, 163 are later expanded; 45 (27.6%) have a subsequently deployed descendant across 20 tasks, and 35 lie on a final deployment's primary-parent path.

For the 45 observed continuations, primary-parent distance to the first deployed descendant is one, two, three, or four edits in 27, 12, three, and three cases. Thus 18 candidates (40.0%) are separated from that descendant by at least two edits. The corresponding generation gaps are one, two, three, or four in 26, 12, four, and three cases. Revisiting an earlier node can skip generations along a single edge. These are candidate-level counts; several candidates can belong to one task or ancestry.

The final path for the Gravitational Wave Detection example in Figure 5a proceeds from root through `b05` (0.0), `c05` (0.0), `d01` (33.3), `e04` (0.0), and `f04` (100.0). Data-to-D3 follows the path `b05` (33.3), `c02` (66.7), `d03` (33.3), `e01` (100.0), and `f01` (100.0). Generation three retains `c04` for deployment while keeping `d03` eligible, then generation four deploys `d03`'s child `e01`. These are birth-generation three-execution means in percent. Both tasks reach 100.0 final Mini reward from an original score of 0.0.

Table 13: Parent eligibility beyond the historical top three. Both searches retain a parent despite three higher-scoring saved versions. Scores are each version's latest complete three-execution mean at the stated generation, in percent. Both tasks improve from 0.0 to 100.0 in the independent final Mini evaluation.

Task	Generation	Retained parent	Higher-scoring saved versions
Gravitational Wave	4	`e04`: 0.0	`e01`: 66.7; `d01`, `b07`: 33.3
Data-to-D3	3	`d03`: 33.3	`c01`, `c02`, `b02`: 66.7

Parent eligibility differs from a global score ranking. Among the saved candidates recorded at each stated generation, both searches retain a parent outside the historical top three (Table 13). Descendants of both retained parents enter the final deployed ancestry.

Explicit information sources outside the final path. Search produces 75 graft candidates: 64 retained and 11 removed as duplicates. Eight retained graft nodes lie on final primary-parent paths across six tasks. Their proposals identify donors outside their tasks' final paths; six donors were not selected for deployment in their own birth generation. Each proposal distinguishes the primary parent from the source of donated material. Appendix I follows the donated procedure through the revised Sales Pivot bundle and its executions.

Depth and edit operators. The 41 adopted lineages contain 126 edits, with 35 lineages having at least two edits and 11 reaching depth five. Mean depth is $126/41 = 3.07$.

Table 14: Adopted lineage depth. The 41 adopted lineages have mean depth 3.07 and median depth three. Depth counts primary-parent edits from the recorded root to deployment.

Adopted lineage depth	Tasks
1	6
2	13
3	5
4	6
5	11

Depth follows primary parent pointers and can be smaller than the generation number when a search revisits an earlier node.

Table 15: Recorded edit operators. Counts cover the 41 experiment trees that produced adopted non-root skills. *Candidates* counts retained candidate versions, *Edits on final paths* counts adopted ancestry edges, and *Final edits* counts edges producing the final deployed bundles. These frequencies do not measure operator success rates.

Operator	Candidates	Edits on final paths	Final edits
consolidate	60	2	2
create	12	3	2
distill	129	13	7
drop	5	0	0
extend	244	39	7
graft	64	8	3
narrow_desc	77	12	6
prune	63	9	3
rewrite	317	40	11

The 971 candidate records and 126 edits on final paths all record Mini as both diagnosis and writing model. Final edits are the 41 edges that produce the final deployed bundles. The name distill denotes a skill-edit operator, not teacher-model distillation.

Tables 14 and 15 summarize complementary aspects of this history. Depth counts successive edits from the recorded search root to the deployed bundle along primary-parent links. Operator counts cover the 971 retained candidates and the edits on adopted paths, excluding the 312 duplicate entries.

The deployed Wyckoff and Exoplanet lineages begin with root children generated after the first generation. The Paratransit and Suricata lineages contain parent links that skip the immediately preceding generation. Primary-parent depth can be smaller than generation number because the search can revisit a bundle from before the immediately preceding generation.

Non-incumbent parents appear on final revision paths. Of the 87 adopted edits generated after generation one, 61 (70.1%) use a parent other than the preceding generation's deployment incumbent, spanning 30 of the 41 adopted lineages (73.2%). Every chosen parent belongs to the eligible set. This final-path statistic has a different denominator from the 289 non-incumbent expansion events above; it includes two later-generation edits whose parent is root.

Among the 29 tasks with improved final Mini reward, 22 (75.9%) use a non-incumbent parent on the final path, accounting for 46 of their 65 post-generation-one edges. The corresponding counts for the 12 tied tasks are eight tasks and 15 of 22 edges. Sixteen of the 29 improved-task final paths pass through an expanded candidate that scored strictly below its birth-generation deployment under the definition above. We use the same 0.05-percentage-point display band as in the endpoint analysis.

Suricata retains `b05` for deployment after the generation-two hold, while reselecting `b01` and root as parents. The reselected `b01` produces the non-incumbent `d05` in generation three, and the final ancestry continues through `e06` and `f01`. The `d05` edit distills a validate-then-compose workflow for HTTP-field rules. Independent final rewards remain 0.0% for Mini and 100.0% for Luna.

Paratransit provides a final-path revisit with a positive final Mini difference (Figure 5c). Generation four retains `d06` for deployment and reselects `d06`, `c05`, and `d04` as parents. Generation-five `f08` is a child of generation-three `d06` and becomes the final selected bundle. The complete path is root

→ b07 → c03 → d06 → f08; d06 also has generation-four children, including e03. Mini's independent final score rises from 0.0 to 33.3, while Luna remains at 0.0.

Later execution evidence guides revisions of earlier bundles. Wyckoff returns to the original bundle after generation two, producing d08 in generation three and final e03 in generation four (Figure 5d). The root-to-d08 edge bypasses two intervening generations, while the edit uses evidence from executions performed after the root bundle was created. Its editing input explicitly quotes the generation-one b06 execution's batch analysis of mounted CIF files. The new skill adds a file-based recipe, Wyckoff-letter grouping, and structured output requirements. In d08's second execution, the agent reads the added guidance and runs a batch analysis over those files. The revision makes the task workflow explicit using the symmetry-analysis method already available in the original reference material. The d08 search rewards are 100.0, 82.0, and 100.0, giving a 94.0 mean. Generation three deploys d04 but retains d08 as a parent, whose child e03 is selected in generation four. Independent evaluation of the complete final e03 bundle raises Mini from 75.7 to 100.0, while Luna's reward remains unchanged at 100.0.

Table 16: **Skill-revision cases and final bundle performance.** Scores show original → evolved reward (%) for complete bundles. These cases illustrate recorded revisions; the scores do not isolate individual edit effects. Evaluation settings follow Section 5.1.

Task	Revision illustrated	Mini	Luna
Software dependency audit	Compare executions; refine report checks, then scan scope	0.0 → 66.7	0.0 → 100.0
PDDL airport planning	Complete a missing helper; strengthen saved-plan checks	33.3 → 100.0	100.0 → 100.0
Sales pivot analysis	Graft PDF extraction from a non-deployed sibling	0.0 → 33.3	0.0 → 100.0
Gravitational wave detection	Consolidate a non-deployed branch's conditioning and search workflow	0.0 → 100.0	100.0 → 100.0

I Concrete Examples of Skill Evolution

These cases connect execution contrasts, proposals, and bundle changes. Table 16 reports representative final task scores. The generation-level scores below are means over three search-time executions, expressed as percentages and distinct from final evaluations. Mini performs execution, diagnosis, and editing throughout these development runs.

I.1 A persistent failure moves the repair upstream

Software Dependency Audit follows root → b06 → c01. Root and b06 each have a mean reward of 66.7%. The failed root execution expands the scan with `--include-dev-deps` and writes 22 findings; the two passing executions use the default production-dependency scope and write three rows. The b06 reporting edit keys vulnerabilities by package, version, CVE identifier, and severity, removes duplicates, and checks the saved CSV against the scan-derived set. Its failed execution deduplicates the 22 rows and verifies their count and key uniqueness, but still expands the scan. Both failures pass the structural tests but fail the content comparison. Checking the report against an expanded scan leaves the required scan scope unverified.

The c01 proposal therefore moves the repair upstream. Its scanning instructions prohibit `--include-dev-deps` while leaving b06's reporting skill unchanged. All three executions use the production scope, write three rows, and check the saved count and key set, yielding 100.0% mean reward. The revision makes explicit the flag's omission from the original command example.

I.2 Workflow consolidation on a retained branch

In Gravitational Wave Detection, generation four deploys `e01` but retains `e04` as a parent (Figure 5a; full path in Appendix H). Although `e04`'s mean reward is 0.0%, each execution passes eight of nine tests and fails only the expected numerical-result comparison. In all three runs, waveform generation and matched filtering use the 15 Hz conditioning cutoff, violating the 20 Hz cutoff specified in the separate search skill.

The `f04` proposal consolidates the two-skill bundle. It moves the search, final write, and readback procedure into the `conditioning` document, then removes the separate `matched-filtering` document. The helper file is unchanged. The merged workflow specifies 15 Hz conditioning alongside the 20 Hz cutoff for waveform generation and matched filtering.

With the consolidated bundle, all three `f04` runs use 20 Hz for search while retaining 15 Hz conditioning. All three executions of each bundle, `e04` and `f04`, write and reread a correctly structured CSV. The `f04` runs pass every test and win deployment with a 100.0% mean reward, compared with 66.7% for `e01`'s fresh batch. Workflow consolidation accompanies a change in the executed computation, and a zero-reward parent produces the next deployed version.

I.3 Helper repair and direct workflow execution

In PDDL Airport Planning, root's search-time mean reward is 0.0%. The generation-one `b06` proposal responds to failed plan syntax by requiring writer-based serialization and parsing the saved plan against the same problem inputs. The initial edit omits its promised helper; validation feedback prompts a repair that adds the executable and its invocation in `SKILL.md`. Although the repaired bundle now includes the helper, the agent cannot locate it in the first `b06` execution. The agent implements the prescribed writing, parsing, and validation workflow directly through the planning library. The saved plan passes official verification.

I.4 A non-deployed sibling contributes an executed procedure

Sales Pivot Analysis separates the value of a bundle for deployment from the value of its individual procedures. Generation one deploys `b04`, which revises spreadsheet construction and has a mean reward of 100.0%. Sibling `b05` revises PDF extraction and has a mean reward of 33.3%. Its first two executions fail pivot-configuration checks while passing state validity. The `c04` proposal selects `b04` as parent and `b05` as donor, citing the donor's successful third execution.

The `c04` bundle contains `b05`'s continuation-aware PDF procedure, including header handling and state filtering, while leaving `b04`'s spreadsheet skill unchanged. Its two passing executions use header-aware extraction and retain full state names. The third converts those names to abbreviations and fails state validity, yielding a search-time mean reward of 66.7%.

Generation two retains `b04` after the historical-score check but selects `c04` for further development. The final bundle descends from `c04`; its primary-parent path therefore includes the revision that imports the procedure from `b05` (Figure 5b).

Procedural compression retains perfect scores. Erlang SSH's generation-one `b09` prunes a 485-line penetration-testing guide to 31 lines focused on reproducing an unauthenticated channel-open attempt and verifying rejection, adds no script, and preserves perfect final scores on Mini and Luna.

J Proposer and Skill Editor Prompt Excerpts

These excerpts from recorded Mini calls for Software Dependency Audit show how the Proposer compares executions and specifies an edit for the Skill Editor. For brevity, some background and runtime details are omitted; operational rules and output formats are retained. Complete templates accompany the source files. Here, `distill` means encoding a procedure extracted from execution evidence in the skill bundle.

Listing 1: Generation-one Proposer excerpt. The prompt provides root execution evidence, instructions for diagnosing failures, and the proposal output format.

```
Differential evolution of a skill library, generation 1 from root
(1-task pool). Evidence: fresh root rollouts below (failures and
behavior).

[ROOT_EXECUTION_EVIDENCE]

## Already proposed this generation (do NOT repeat or near-duplicate)
[PREVIOUS_PROPOSALS]

## Proposals REJECTED so far and why (each cost a round and produced no
child)
[REJECTION_FEEDBACK]

## Allowed operators (one per child)
prune / rewrite / extend / create / consolidate. Every one of these
writes a COMPLETE SKILL.md, so any of them can change the description,
the body, or both, and, where the scripts section below appears, the
script-capable ones can ship a file under scripts/ in the same child.
Choose by what the evidence says is broken; there is no default
operator. create ONLY if no host skill exists: narrow description naming
a concrete task domain, max 2 per generation.
consolidate MERGES: the target skill survives under its own name and
every OTHER skill in the library is folded into it and its directory
deleted. The merged description must still fire wherever each folded
skill fired -- it is rejected if it drops the trigger terms no surviving
skill carries -- and the merged body must teach each folded procedure in
one order an agent can follow, not concatenate them. Only these can
survive a fold here: cvss-score-extraction,
trivy-offline-vulnerability-scanning, vulnerability-csv-reporting.
A child may also ship EXECUTABLE files: rewrite, extend, create,
consolidate may add new files under the target skill's scripts/, and
every one of them is run inside this task's own container before the
child is accepted. Aim a direction there where the evidence shows the
agent doing something EVERY RUN that a program could do once and better:
deriving a value it cannot look up, or confirming a result it cannot
otherwise observe before submitting. Prose cannot supply the first and
does not help with the second. A helper that only restates the text can
merely break. Shipping one means writing the file AND the sentence that
makes it reachable; those are one edit, not two.

## Existing skills
[EXISTING_SKILL_NAMES]

## Consistency review (advisory, not a content lock)
An unmet grader noun is not an edit direction.  If a grader says an
object is invalid,
missing, extra, or unreachable, first use the visible run state to
decide whether the
losing run added it, omitted it, or transformed it wrongly.  Do not turn
the noun alone
into an instruction to add or preserve that object.

Agent-visible executable source and a state/action from a run that
PASSED are consistency evidence which the diagnosis must reconcile.
They are not immutable content: a proposal may replace existing behavior
when the failing/passing contrast explains why and states the narrower
invariant that should replace it.  For each proposal, quote the relevant
executable line when the target skill has one in the evidence block, and
quote a concrete successful state/action when a passing run is shown.  A
recovered exception, a command line by itself, or "Process exited with
code 0" is not successful-state evidence.  Then state specifically
whether the direction preserves, narrows, or deliberately replaces the
quoted behavior and why that is consistent with the measured contrast.
These fields are retained for audit.  Missing or imperfect quotation is
advisory and never discards a proposal; official evaluation remains the
```

```
only arbiter of the edit.  Use NONE only when the corresponding kind of
evidence is genuinely absent.

## How to read the evidence, decide, and rank

How to read the evidence. The grader's verdict blocks above are WHAT
decides the score: a run scores zero because a requirement was still
standing when it ended, and each requirement's shown / lost / survived
counts are beside it. The rollout blocks are WHY -- that is where you do
the actual thinking. They are long and full of things going wrong, and
most of it is the agent finding its own errors and fixing them; a block
tagged as coming from a replicate that PASSED is proof its errors were
recoverable. An error a run recovered from, or a requirement that also
appears in runs that passed, is not what loses the task, however often
it appears.

A cause has to EXPLAIN the verdict, not appear in it. A cause you cannot
trace to the verdict at all is a detour; one the verdict never names but
that explains it is usually the real one, and you reach it only through
the runs.

The verdict is for YOU, not for the agent. The agent never sees the
verifier; it grades after the agent has finished. So a direction must be
actionable on what the agent CAN observe before it finishes: the source
it was given, the files in its working tree, the output of a command it
can run. And the verdict LOCATES the problem; it is not an answer to
hand over. An edit that transcribes the names the grader mentions has
cached this instance's answer -- the agent can already reach them. Aim
the direction at how to OBTAIN the required surface.

What you are deciding is what would make this task succeed RELIABLY
rather than sometimes. You have root's replicates above; the writer
downstream sees one direction and the file it edits, so the judgement
has to be made here and stated in the direction, specifically enough
that someone who cannot see these runs could carry it out exactly. So
the other half of every finding is the path that WORKED: at the point a
failing run went wrong, what did a passing run do instead? Where the
replicates contain a pass/fail pair, that is the cleanest source -- the
library text was identical, so whatever differs is the path. Quote it in
what_passing_runs_did; "no run has ever got past this" is a real answer,
and it labels the edit an untested guess, which is worth knowing.

Rank the causes before you propose. State each cause as a SENTENCE
naming a mechanism. Rank by how many replicates LOST because of it, not
how many show it, and report both plus how many showed it and passed
anyway. If a cause is outside what any skill edit could address, still
rank it honestly and mark it "skill_can_address": false -- the attack
quota below then applies to the highest cause a skill CAN address.

One signature deserves priority whenever it appears: the GRADED VALUE
clusters. When the number the grader checks comes out in a few tight
groups across runs of the SAME library -- within a group the values
agree to far more precision than chance would allow -- that is not
noise: the runs took different deterministic computation paths at some
step, and which path a run takes decides the score. Diff a run from each
cluster at the step that produces the graded number, name the divergence
as a cause, and propose pinning the passing path as an explicit
procedure in the skill -- the exact ordering/normalization/formula, not
the target value. Rank it through the same evidence rules as every other
cause.

At least 2 of your 6 proposals must attack the first-ranked addressable
cause (causes tied at the top score count as jointly first) and at least
3 must attack the top two; the rest are free, and that is where a small
or odd observation belongs.
```

```
Before committing to each proposal, check it against the runs: the cause
accounts for the runs that lost; it is absent from, or visibly overcome
in, the runs that passed; and the direction is executable by someone who
cannot see these runs. A proposal that fails any of these wastes its
child.

## Output strict JSON only
{"loses_the_task": [{"cause": "<one sentence naming the MECHANISM that
loses the task; it must EXPLAIN the grader's complaint and may name
things the verdict never mentions>", "skill_can_address": <false ONLY if
no skill edit could act on this cause; omit otherwise>, "runs_showing":
<how many runs show it at all>, "runs_affected": <of those, how many
LOST because of it -- a run that showed it and still passed does not
count>, "also_in_passing_runs": <how many showed it and passed anyway; 0
is a strong cause, a large number means you are looking at a symptom>,
"what_passing_runs_did": "<the step a run that PASSED took at this same
point, quoted from its trace -- the command, the line it wrote, the
order it did things in. "no run has ever got past this" is a real answer
and says the fix is not yet in the evidence>", "evidence_quote":
"<verbatim>"}, ...],
 "diagnoses": [{"parent": "root", "operator":
"prune|rewrite|extend|create|consolidate",
  "target_skill": "<name>", "direction": "<one-line edit>",
  "problem": "<what the evidence shows>",
  "failing_test": "<the test that fails in most replicates, by name>",
  "diverged_at": "<the step in the FAILING run that produced what that
test rejects -- the command it ran or the file it wrote, quoted from the
trace. If no run ever opened this skill there is no such step: say so,
and that absence is itself the finding>",
  "would_change": "<what the agent does differently at that step, and
whether it could have known there that it was wrong -- if it could not,
say what makes it observable>",
  "executable_consistency_quote": "<verbatim tagged line from this
target skill's AGENT-VISIBLE EXECUTABLE block, or NONE only if absent>",
  "passing_consistency_quote": "<verbatim tagged successful state/action
from a passing run, or NONE only if absent>",
  "consistency_check": "<whether this direction preserves, narrows, or
replaces the quoted behavior, and why the measured pass/fail contrast
supports that relationship>",
  "ships_file": "<true if this edit delivers an executable file. Say
true whenever the cause is that runs submit something they never checked
-- an instruction can only say what to do, a check reports whether it
was done. The contract is that a declaration means a file: say true and
the writer must emit it in a ===SCRIPT block, and prose describing a
file is not the file>",
  "attacks": "<the exact `cause` string from loses_the_task this edit is
aimed at>",
    "evidence_quote": "<verbatim from evidence above>"}, ...]}
They are asked for because an edit that cannot say where in a run it
acts has changed a topic and not a run. Answer them from the trace, and
answer them honestly -- "no run opened this skill, so there is no such
step" is a real answer and points at the description; an invented step
is worth less than none.
Exactly 6 diagnoses, diverse targets and operators. [...]
```

Listing 2: Later-generation Proposer excerpt. The prompt supplies candidate comparisons, parent and donor constraints, and the proposal output format.

```
Differential evolution of a skill library, generation 2. Breeding
parents: b06, b01, b04 (lineages of root + accumulated single edits).
Evidence below: grader verdicts, the previous generation's population
contrast, and raw run traces.
```

```
[POPULATION_EXECUTION_EVIDENCE]

## Already proposed this generation (do NOT repeat these or
near-duplicates)
[PREVIOUS_PROPOSALS]

## Edits already MEASURED as net losses (closely similar directions --
same operator and target -- are auto-rejected)
[PREVIOUSLY_MEASURED_LOSING_EDITS]

## Proposals REJECTED so far and why (each cost a round and produced no
child)
[REJECTION_FEEDBACK]

## Allowed operators (one per child)
prune / rewrite / graft / extend / create / drop / narrow_desc /
consolidate. Every one of these except narrow_desc and drop writes a
COMPLETE SKILL.md, so any of them can change the description, the body,
or both, and rewrite / graft / extend / create / distill / consolidate
can also ship a file under scripts/ in the same child -- writing that
file and the sentence that makes it reachable is ONE edit, not two.
Choose by what the evidence says is broken; there is no default and no
preferred operator. graft may source a section from ANY of b06, b01, b04
into a breeding parent (name source_node). create ONLY if no host
exists: narrow description naming a concrete task domain, max 2 per
generation.
distill: encode the method a winning run actually used. YOU are the one
reading those runs, so state the method itself in `direction` and
`would_change` -- the steps in order, the commands or checks that
mattered, what the losing runs did instead. The editor places what you
write; it does not see the runs and cannot re-derive them.
DELIVERY MECHANISM: before the task instruction the agent is shown EVERY
mounted skill as name + full description, permanently, and is told that
a task clearly matching a skill's description MUST use that skill.
Nothing is hidden behind a directory listing. The description decides
whether the body is ever opened; the body decides what the agent DOES --
and the body delivers in two forms, not one. It can INSTRUCT, which the
agent then follows as well as it happens to, or it can SHIP A FILE UNDER
scripts/ and name the command that runs it, which reports. An
instruction can only say what to do; a check says whether it was done.
Where a cause is that runs submit something unverified, the file is the
stronger edit and the operators rewrite / extend / graft / distill /
consolidate / create can all carry one. This library holds 3 skills:
nothing is buried and there is no sibling to win attention from, so the
only useful description edit makes the skill fire more reliably --
ENUMERATE the concrete triggers of the task it serves, in that task's
own vocabulary. Taking trigger surface away (narrowing, disclaiming what
the skill is not for) has no competitor to beat here and can only stop
the skill firing. The leverage is in the body's procedure.
drop REMOVES one whole skill from the library -- propose it where the
evidence shows a skill actively misleads the agent on this task; the
removal is the child's entire edit. narrow_desc rewrites ONLY a skill's
description line, leaving the body byte-identical; what that line should
then say is set by the DELIVERY MECHANISM paragraph above, which depends
on how many skills this library holds.
consolidate MERGES THE LIBRARY DOWN: the target skill survives under its
own name and every OTHER skill of that parent is folded into it and its
directory deleted. Both halves are checked mechanically, so propose it
only where they hold: the merged description must still fire wherever
each folded skill fired (it is rejected if it drops the trigger terms no
surviving skill carries), and the merged body must teach each folded
procedure in one order an agent can follow rather than concatenating
them. Which skill may survive a fold, by parent -- b06, b01, b04:
```

```
cvss-score-extraction, trivy-offline-vulnerability-scanning,
vulnerability-csv-reporting.
A child may also ship EXECUTABLE files: rewrite, graft, extend, create,
distill, consolidate may add new files under the target skill's
scripts/, and every one of them is run inside this task's own container
before the child is accepted. Aim a direction there where the evidence
shows the agent doing something EVERY RUN that a program could do once
and better: deriving a value it cannot look up, or confirming a result
it cannot otherwise observe before submitting. Prose cannot supply the
first and does not help with the second. A helper that only restates the
skill's prose adds risk without adding information. Shipping one means
writing the file AND the sentence that makes it reachable; those are one
edit, not two.

## Existing skills
[EXISTING_SKILL_NAMES]

## Consistency review (advisory, not a content lock)
An unmet grader noun is not an edit direction.  If a grader says an
object is invalid,
missing, extra, or unreachable, first use the visible run state to
decide whether the
losing run added it, omitted it, or transformed it wrongly.  Do not turn
the noun alone
into an instruction to add or preserve that object.

Agent-visible executable source and a state/action from a run that
PASSED are consistency evidence which the diagnosis must reconcile.
They are not immutable content: a proposal may replace existing behavior
when the failing/passing contrast explains why and states the narrower
invariant that should replace it.  For each proposal, quote the relevant
executable line when the target skill has one in the evidence block, and
quote a concrete successful state/action when a passing run is shown.  A
recovered exception, a command line by itself, or "Process exited with
code 0" is not successful-state evidence.  Then state specifically
whether the direction preserves, narrows, or deliberately replaces the
quoted behavior and why that is consistent with the measured contrast.
These fields are retained for audit.  Missing or imperfect quotation is
advisory and never discards a proposal; official evaluation remains the
only arbiter of the edit.  Use NONE only when the corresponding kind of
evidence is genuinely absent.

## Rank the causes BEFORE you propose anything

[PREVIOUS_GENERATION_LEADING_CAUSE]
Its evidence is STILL PRESENT in the block above. Rank it again, or
state as its own entry why it is no longer what loses this task -- an
edit landed and worked, or the runs moved on. Dropping it silently is
the one thing that is not an answer.

How to read the evidence. The grader's verdict (the first block above)
is WHAT decides the score: a run scores zero because a requirement was
still standing when it ended, and each requirement's shown / lost /
survived counts are beside it. The trajectories are WHY -- that is where
you do the actual thinking. They are long and full of things going
wrong, and most of it is the agent finding its own errors and fixing
them; an error a run recovered from, or a requirement that also appears
in runs that PASSED, is not what loses the task, however often it
appears.

A cause has to EXPLAIN the verdict, not appear in it. A cause you cannot
trace to the verdict at all is a detour; one the verdict never names but
that explains it is usually the real one, and you reach it only through
the runs.
```

```
The verdict is for YOU, not for the agent. The agent never sees the
verifier; it grades after the agent has finished. So a direction must be
actionable on what the agent CAN observe before it finishes: the source
it was given, the files in its working tree, the output of a command it
can run. And the verdict LOCATES the problem; it is not an answer to
hand over. An edit that transcribes the names the grader mentions has
cached this instance's answer -- the agent can already reach them. Aim
the direction at how to OBTAIN the required surface.

What you are deciding is what would make this task succeed RELIABLY
rather than sometimes. You have the replicates and contrasts above; the
writer downstream sees one direction and the file it edits, so the
judgement has to be made here and stated in the direction, specifically
enough that someone who cannot see these runs could carry it out
exactly. So the other half of every finding is the path that WORKED: at
the point a failing run went wrong, what did a passing run do instead?
Where the evidence contains a same-library pass/fail pair, that is the
cleanest source -- the text was identical, so whatever differs is the
path. Quote it in what_passing_runs_did; "no run has ever got past this"
is a real answer, and it labels the edit an untested guess, which is
worth knowing.

Rank the causes before you propose. State each cause as a SENTENCE
naming a mechanism. Rank by how many runs LOST because of it, not how
many show it, and report both plus how many showed it and passed anyway.
If a cause is outside what any skill edit could address, still rank it
honestly and mark it "skill_can_address": false -- the attack quota
below then applies to the highest cause a skill CAN address.

One signature deserves priority whenever it appears: the GRADED VALUE
clusters. When the number the grader checks comes out in a few tight
groups across runs of the SAME library -- within a group the values
agree to far more precision than chance would allow -- that is not
noise: the runs took different deterministic computation paths at some
step, and which path a run takes decides the score. Diff a run from each
cluster at the step that produces the graded number, name the divergence
as a cause, and propose pinning the passing path as an explicit
procedure in the skill -- the exact ordering/normalization/formula, not
the target value. Rank it through the same evidence rules as every other
cause.

Being DRAWN to a vivid failure is fine and often useful; being drawn
entirely to it is not. At least 2 of your 6 proposals must attack the
first-ranked addressable cause (causes tied at the top score count as
jointly first) and at least 3 must attack the top two; the rest are
free, and that is where a small or odd observation belongs.

Before committing to each proposal, check it against the runs: the cause
accounts for the runs that lost; it is absent from, or visibly overcome
in, the runs that passed; and the direction is executable by someone who
cannot see these runs. A proposal that fails any of these wastes its
child.

## Output strict JSON only
{"loses_the_task": [{"cause": "<one sentence naming the MECHANISM that
loses the task; it must EXPLAIN the grader's complaint and may name
things the verdict never mentions>", "skill_can_address": <false ONLY if
no skill edit could act on this cause; omit otherwise>, "runs_showing":
<how many runs show it at all>, "runs_affected": <of those, how many
LOST because of it -- a run that showed it and still passed does not
count>, "also_in_passing_runs": <how many runs showed it and passed
anyway; 0 is a strong cause, a large number means you are looking at a
symptom>, "what_passing_runs_did": "<the step a run that PASSED took at
```

```
this same point, quoted from its trace -- the command, the line it
wrote, the order it did things in. "no run has ever got past this" is a
real answer and says the fix is not yet in the evidence>",
"evidence_quote": "<verbatim>"}, ...],
 "diagnoses": [{"parent": "b06|b01|b04", "operator":
"prune|rewrite|graft|extend|create|drop|narrow_desc|distill|consolidate",
  "source_skill": "<for graft ONLY, optional: donor SKILL name when
transplanting a section into a different skill>",
  "target_skill": "<name>", "source_node": "<for graft ONLY: the LIBRARY
id to copy the section FROM, one of the comparison set above - NOT a
skill name; omit otherwise>",
  "task": "<OPTIONAL: the pool task this edit aims to improve, exactly
as named in the evidence; when given, the editor receives that task's
instruction text>",
  "direction": "<one-line edit>", "problem": "<what the evidence shows>",
  "failing_test": "<the test that fails in most replicates, by name>",
  "diverged_at": "<the step in the FAILING run that produced what that
test rejects -- the command it ran or the file it wrote, quoted from the
trace. If no run ever opened this skill there is no such step: say so,
and that absence is itself the finding>",
  "would_change": "<what the agent does differently at that step, and
whether it could have known there that it was wrong -- if it could not,
say what makes it observable>",
  "executable_consistency_quote": "<verbatim tagged line from this
target skill's AGENT-VISIBLE EXECUTABLE block, or NONE only if absent>",
  "passing_consistency_quote": "<verbatim tagged successful state/action
from a passing run, or NONE only if absent>",
  "consistency_check": "<whether this direction preserves, narrows, or
replaces the quoted behavior, and why the measured pass/fail contrast
supports that relationship>",
  "ships_file": "<true if this edit delivers an executable file. Say
true whenever the cause is that runs submit something they never checked
-- an instruction can only say what to do, a check reports whether it
was done. The contract is that a declaration means a file: say true and
the writer must emit it in a ===SCRIPT block, and prose describing a
file is not the file>",
  "attacks": "<the exact `cause` string from loses_the_task that this
edit is aimed at>",
    "evidence_quote": "<verbatim from evidence above>"}, ...]}
They are asked for because an edit that cannot say where in a run it
acts has changed a topic and not a run. Answer them from the trace, and
answer them honestly -- "no run opened this skill, so there is no such
step" is a real answer and points at the description; an invented step
is worth less than none.
Exactly 6 diagnoses, diverse targets and operators, each grounded in the
evidence above. [...]
```

J.1 SKILL EDITING FROM A PROPOSAL

The Skill Editor receives the proposal, parent skill, and task context. The excerpt below illustrates a rewrite call; the current skill is represented by an input slot.

```
Apply ONE rewrite edit to an agent skill (SKILL.md with YAML frontmatter
name/description + markdown body) used by a frozen CLI coding agent on
these tasks -- software-dependency-audit.

## The edit to make -- a specification to execute, not text to copy
{
  "operator": "rewrite",
  "target_skill": "trivy-offline-vulnerability-scanning",
  "direction": "Rewrite the scan instructions to keep this audit on
`/root/package-lock.json` with the default offline production-dependency
scope and explicitly reject `--include-dev-deps`.",
```

```
  "problem": "Some runs broaden the vulnerability universe and write a
dev-inclusive CSV that does not match the grader's expected set.",
  "failing_test":
"test_outputs.py::TestSecurityAuditTask::test_csv_matches_ground_truth",
  "diverged_at": "trivy fs /root/package-lock.json --format json
--output /root/trivy_report_dev.json --scanners vuln --skip-db-update
--offline-scan --include-dev-deps --cache-dir /root/.cache/trivy",
  "what_should_happen_there": "At the scan command, the flag choice is
directly observable, so the agent should omit the dev-deps flag and
continue only with the production-only report.",
  "decided_from_this_line_in_the_runs": "trivy fs
/root/package-lock.json --format json --output
/root/trivy_report_dev.json --scanners vuln --skip-db-update
--offline-scan --include-dev-deps --cache-dir /root/.cache/trivy",
  "executable_consistency_evidence": "[...]",
  "successful_state_consistency_evidence": "trivy fs
/root/package-lock.json --scanners vuln --offline-scan --skip-db-update
--cache-dir /root/.cache/trivy --format json -o /root/trivy_report.json",
  "diagnosis_consistency_check": "Narrows the command path to the
observed passing scan while preserving the offline Trivy workflow."
}

Objective: this edit is judged on one thing -- whether the frozen agent
passes this task's own tests more often, repeatably across independent
runs rather than once by luck. Wording that does not change what the
agent DOES scores nothing.
Your reader is an agent doing this task. The diagnosis upstream read the
runs and decided what that agent needs; the decision is the direction
above, and your job is to make it real in the file. The form is yours --
a procedure, a rule, a check it runs, a fact it was missing -- except
where the direction itself names one.
When the specification contains executable_consistency_evidence or
successful_state_consistency_evidence, reconcile the completed edit with
them. They are evidence, not frozen content: preserve them when they
describe the working invariant, or deliberately revise them only where
the supplied diagnosis_consistency_check explains the failing/passing
boundary.
A skill reaches the agent in two layers. The DESCRIPTION is always in
the agent's context and decides whether the body is ever opened. The
BODY then delivers one of two things: instructions, which the agent
follows as well as it happens to, or a file it runs, which can report
whether the work was actually done. The direction and problem above tell
you which layer this edit acts on.
Three rules for what goes in the body:
- The agent NEVER sees the verifier; it grades after the agent has
finished. Never write anything triggered by grader output ('if the
verifier reports X...'). Write triggers the agent can observe before
finishing: the source it was given, the files in its working directory,
the output of a command it can run itself. Stating a requirement as a
fact about the task is fine.
- Teach how to OBTAIN what the task requires, not what it is. 'Read the
spec the task mounts and match the names it imports before writing the
implementation' transfers; a table enumerating this task's own names is
a cached answer that transfers nowhere. One or two names as an example
of the shape is fine.
- Every line must act on the cause the direction names. General
procedure is welcome when it does; material that is merely about the
same subject, however correct, is padding. Test each line: at the step
where runs were losing, does an agent that has read it do something
different?
The agent cannot edit the skill and cannot see this prompt, so never
write body text telling its reader to name a script or amend the skill;
if a script should run, ship it and give the command that runs it.
Place the change where the agent will be standing when it matters. If,
at that point, the agent could not have known it was about to go wrong,
```

```
no added sentence will move it -- what changes the run is something that
makes the wrongness observable there: a file it runs whose output it
must read before going on.
Carry the direction out in full: a file named means the file shipped and
named at the point the agent runs it; a step named means the step placed
where the agent will reach it; a removal named means it is gone. Before
returning, read the file as the agent will: every path, command and name
in it must exist; nothing you added may contradict what you kept; if you
would not follow it yourself, it is not done.

ONE EDIT IS ONE INTENT, not one file. If the change you decide on needs
more than one part to exist at all -- a new script is unreachable until
a sentence names it, a moved section is dangling until its pointer moves
-- those parts ARE the edit, and shipping them together is not two
edits. What "one" forbids is two unrelated intents in the same child,
because then no measurement can say which of them paid.

RETURN THE REST OF THE FILE BYTE FOR BYTE. You are asked for the whole
SKILL.md because that is the only way to receive a change, not because
the whole file is in scope. Lines you are not editing come back exactly
as they arrived -- same bold markers, same quote characters, same
dashes, same wrapping. Retyping them costs something real: the reader of
this lineage cannot see which lines carry the intent, the duplicate
check reads a large change where there was a small one, and the
measurement that follows cannot attribute what it measures.

Rules: the response starts with the complete new SKILL.md, beginning
with ---. frontmatter name: must be exactly
"trivy-offline-vulnerability-scanning". The description is the ROUTING
LAYER: every skill's description sits in the agent's context
permanently, while bodies are opened only when a description wins the
agent's attention. This library holds 3 skills, so routing pressure is
low -- there is little sibling overlap to remove and the agent has few
descriptions to choose between, so the body's procedure is usually where
the leverage is. Which layer THIS edit acts on is settled by the
direction, not by that tendency. Never widen the description into a
claim the body cannot serve. ASCII only, no emoji, at most 315 lines
(the current version is 243); do not delete material gratuitously -- but
material the direction or problem identifies as unused or misleading may
go. Preserve what still serves the task.

<<<TASK-INSTRUCTION software-dependency-audit
## the task instruction for software-dependency-audit -- the agent reads
this in full at every run
The agent already has every line of it. Repeating any of it in the skill
changes nothing the agent does; contradicting it (a different output
path, a different library version) costs the run.
You are a software security engineer. Given a dependency file, you need
to perform a security audit to identify vulnerabilities in third-party
dependencies.

The dependency file is given in `/root/package-lock.json`.

You can use offline tools or database.

Only detect the vulnerabilities with severity levels of HIGH and
CRITICAL.

For each vulnerability, collect the following information:
- Package name
- Installed version
- CVE ID
- Severity level
- CVSS score (e.g. from NVD, GHSA, or RedHat)
- Fixed version (if available; if not available, leave it N/A)
```

```
- Vulnerability title/description
- Reference URL

Write the results to `/root/security_audit.csv` with the following
columns as
`Package,Version,CVE_ID,Severity,CVSS_Score,Fixed_Version,Title,Url`
>>>END-TASK-INSTRUCTION

## Measured routing fact (15 rollout(s); opened in 14 of 15)
14 of 15 previous rollout(s) opened
trivy-offline-vulnerability-scanning/SKILL.md, and none of them opened
any file under its references/, scripts/ or assets/.

## Trigger surface (the description decides whether this body is ever
opened)
The task's own vocabulary is: audit, dependencies, dependency, engineer,
file, given, identify, party, perform, security, software, third,
vulnerabilities. A description that DISCLAIMS one of these stops the
skill firing on the task it exists for; excluding a task the skill does
help costs score directly, and excluding one it does not help gains
nothing unless a sibling skill is being confused with it.
trivy-offline-vulnerability-scanning shares at most 5 distinctive
term(s) with any of the 2 other skill(s) in this library.

## Shipping an executable helper
You may append one or more files after the SKILL.md, each introduced by
its own line:

===SCRIPT scripts/<filename>===
<complete file content>

Gates, each enforced mechanically:
- at most 4 files, all under scripts/, 40000 bytes total
- the SKILL.md must NAME every file you ship, at the point in the
procedure where the agent should run it; an unnamed file is unreachable
and the child is rejected
- every file is EXECUTED inside this task's own container before the
child is accepted: its module level must run to completion and it must
survive being run with no arguments -- import only what that container
image already has, and stop cleanly when inputs are missing rather than
assuming them
- a direction that names an executable, answered with prose alone, has
not made the edit; deliver the file
A file the skill ALREADY ships is changed by re-emitting the whole file
with your change in it; keep everything that still works -- the diff
against the inherited version is recorded. Change one only where the
direction says that file is what is wrong.
When the direction does not name a file you may still ship one, and
where the cause is that runs submit something unchecked, you usually
should: an instruction can only say what to DO, while a check the agent
runs can say whether it was DONE, and a check is followed every time
while a sentence is followed as well as it happens to be. What a file is
FOR: to derive a value the agent has no way to look up, or to let it
OBSERVE a result it otherwise submits unchecked. Not for decoration --
ship one that acts on the cause, not one that demonstrates the
capability. The file and the sentence that makes it reachable are one
edit (see ONE EDIT IS ONE INTENT above).

## Current version
[CURRENT SKILL DOCUMENT]

## What to output, now that you have read
trivy-offline-vulnerability-scanning
  - the COMPLETE new SKILL.md, starting with ---
```

```
  - you MAY append files after it, each introduced by a line ===SCRIPT
scripts/<filename>===, and should where the direction or the cause is
that runs submit something unchecked
  - make this one edit and no other: Rewrite the scan instructions to
keep this audit on `/root/package-lock.json` with the default offline
production-dependency scope and explicitly reject `--include-dev-deps`.
  - every line you add has to change what the agent DOES at the step
where runs were losing; material that reads like the rest of this file
but does not act on that is padding
```