SKILL^2: SKILL EVOLUTION AS POLICY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Skills are increasingly generated and refined automatically, but the dominant sequential-refinement paradigm—execute the current skill, reflect on its rollout, and rewrite the whole skill—often degrades over rounds. We trace this failure to three missing ingredients from reinforcement learning: a baseline for the return, credit assignment to individual design choices, and a replay buffer that preserves evaluated skills and trajectories. These mechanisms are not optional heuristics: without a return baseline, dimension-level credit assignment, and a replay buffer, the effect of any design choice is unidentifiable from a single rollout. We therefore reframe skill evolution as optimizing the policy that writes skills, not the skill document itself. The actions are discrete design choices within a skill, and the policy parameter is a growing context of evaluated skills, trajectories, returns, and conclusions. We propose Skill^2, which performs controlled one-dimension-at-a-time comparisons: alternatives in a group differ only in the chosen dimension, so the rest of the group serves as a baseline; the return gap is credited to that dimension; and every evaluated skill remains available as a reusable control. The final skill is synthesized from choices that won such comparisons, while unsupported or null dimensions are removed. On SkillLearnBench, across three LLMs, Skill^2 achieves 52.0% average accuracy, 9.4 points above the no-skill agent and 11.1 points above the strongest skill-generation baseline, and improves held-out transfer and judged skill completeness/determinism. Ablations confirm the contribution of controlled comparison, control reuse, and synthesis.

1 INTRODUCTION

LLM agents are increasingly deployed on specialized, long-horizon tasks that require procedural knowledge the base model lacks (Anthropic, 2025a; OpenAI, 2025). *Skills* supply such knowledge. A skill is a package of instructions, scripts, and resources that the agent loads on demand, extending its capabilities without changing the weights (Anthropic, 2025b; Xu & Yan, 2026; Li et al., 2026a). Human-authored skills, however, require expert effort for every workflow and go stale as tasks and tools change. *Skill evolution* has recently emerged in response. It lets a model generate a skill and improve it from the agent's execution feedback (Anthropic, 2025c; Ni et al., 2026; Yang et al., 2026; Zhong et al., 2026).

Skill evolution is commonly realized as *sequential refinement*, in which an LLM reflects on the agent's execution and rewrites the skill round by round (Alzubi et al., 2026; Zhang et al., 2026b; Zheng et al., 2025). Following TextGrad (Yuksekgonul et al., 2024), SkillGrad and SkillOpt treat the skill as the parameter to optimize and use this reflection in place of a gradient (Wang et al., 2026b; Yang et al., 2026; Shen et al., 2026). On the realistic multi-step tasks of SkillLearnBench (Zhong et al., 2026), however, sequential refinement degrades as rounds accumulate (Figure 1(a)). The difficulty lies in treating the skill as the optimization target. A skill consists of modules, each implementing one of several alternative designs, such as the vulnerability scanner in Figure 1(b). In a single reflection, the LLM can hardly tell which of these designs caused the outcome. When it credits the wrong design, the next round builds on that mistake, and such mistakes accumulate as refinement continues.

Rather than optimizing the skill itself, we optimize the policy that writes skills. Under this view, each design choice is an action, a skill is the sequence of actions the policy composes, and the return of executing the skill is what the policy learns to maximize (Figure 1(c)). Skill evolution thus becomes policy optimization, a problem that reinforcement learning has long studied. Three of its principles

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

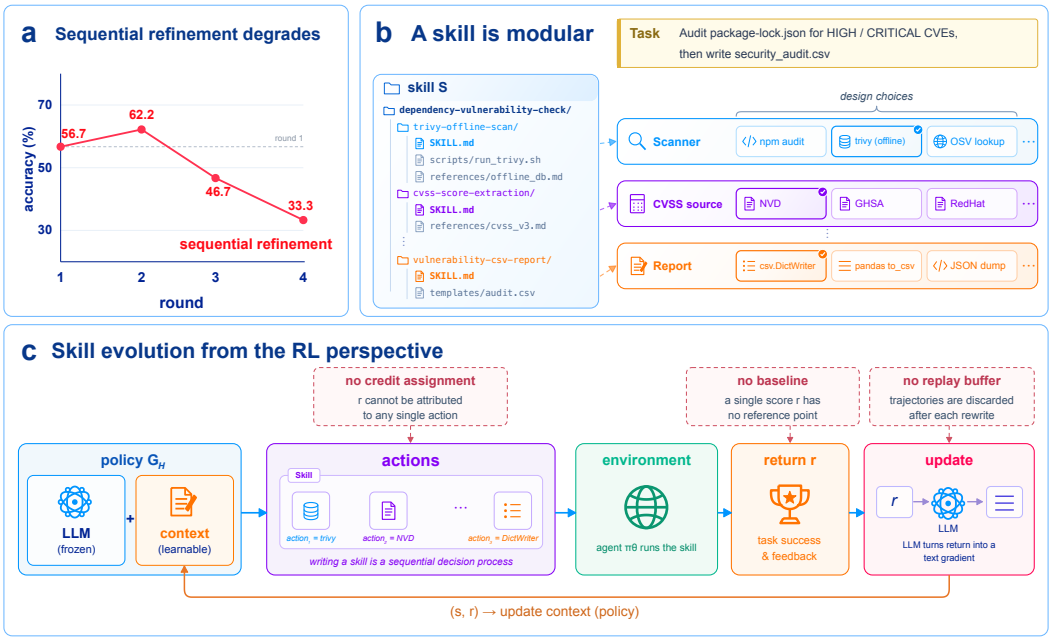


Figure 1: (a) SELF-FEEDBACK accuracy over rounds (SkillLearnBench, Claude Sonnet 4.6) (Zhong et al., 2026). (b) Skill modules with alternative designs. (c) Skill evolution as policy optimization.

bear directly on the difficulty above. A baseline gives a return a reference point, credit assignment attributes the return to the actions responsible for it, and a replay buffer stores past experience so that each costly execution keeps informing later updates. Sequential refinement has none of the three. It judges each return in isolation, credits it to the skill as a whole, and discards each trajectory once the skill is rewritten.

We propose **Skill**^{2}, which brings these three principles into skill evolution. Instead of rewriting a skill as a whole, Skill^2 compares skills that differ in one choice alone. The alternatives then serve as the baseline, and the difference in return is credited to that choice. The resulting conclusions accumulate in the context of the skill generator together with the skills and trajectories behind them, and later comparisons reuse this experience. The final skill is written from what has been learned about each choice.

We evaluate Skill^2 on SkillLearnBench (Zhong et al., 2026), a benchmark of realistic skill-dependent tasks executed in containerized environments. Across three LLMs, Skill^2 attains 52.0% average accuracy, 9.4 points above the agent without skills and 11.1 points above the strongest skill-generation baseline, which itself does not improve on the agent without skills.

Our contributions are:

1. We reframe skill evolution as optimizing the policy that writes skills rather than the skill itself, with individual design choices as actions (Sections 3.1–3.2), and trace the degradation of sequential refinement to the lack of a baseline, of credit assignment, and of a replay buffer.

2. We propose Skill^2 (Section 3.3), which performs counterfactual credit assignment over design choices, estimating the effect of each choice against alternatives that differ in it alone, and reuses every evaluated skill retained in the policy's context.

3. Experiments on SkillLearnBench (Section 4) show that Skill^2 outperforms all automated baselines for each of three LLMs, exceeding the strongest skill-generation baseline by 11.1 points on average.

2 Related Work

Agent skills. A skill is a package of instructions, scripts, and resources that an agent loads on demand to extend its capabilities without retraining (Anthropic, 2025b; Xu & Yan, 2026). Beyond manual authoring, several lines of work acquire skills automatically. Trace2Skill and SkillX distill skills from large pools of trajectories (Ni et al., 2026; Wang et al., 2026a). Memento-Skills and SkillOS manage a growing library, deciding which skills to store, route, and retire (Zhou et al., 2026; Ouyang et al., 2026). Other methods train the agent weights together with a skill bank (Xia et al., 2026; Li et al., 2026b; Tu et al., 2026; Lu et al., 2026). Benchmarks measure what skills contribute. SkillsBench and SWE-Skills-Bench compare paired runs with and without a curated skill (Li et al., 2026a; Han et al., 2026), and Liu et al. (2026b) find that when agents must retrieve skills from 34k real-world skills instead of receiving curated ones, pass rates fall close to the no-skill baseline. SkillLearnBench (Zhong et al., 2026) evaluates skills that a method generates from seed instances, scoring skill quality and task outcomes on held-out instances of realistic multi-step tasks, and SEA-Eval (Jiang et al., 2026) follows agents that evolve across a task stream.

Skill evolution. Most skill evolution methods follow sequential refinement. The frozen agent executes the current skill, an LLM reflects on the resulting trajectories, and the skill is rewritten before the next round. Methods differ mainly in the feedback that drives the rewrite. EvoSkill and SkillForge diagnose execution failures and trace them back to deficiencies in the skill (Alzubi et al., 2026; Liu et al., 2026a), CoEvoSkills obtains feedback from a co-evolving verifier when no ground truth is available (Zhang et al., 2026b), SkillClaw aggregates trajectories from many users into collective updates (Ma et al., 2026), and SkillOpt, SkillGrad, and SkillOpt-Lite express the reflection as a textual gradient on the skill (Yang et al., 2026; Wang et al., 2026b; Shen et al., 2026). To stabilize the rewrite, SkillOpt bounds its size with an edit budget (Yang et al., 2026), and Skill-Pro and Skills-Coach accept a revised skill only after it outperforms the previous one (Mi et al., 2026; Tian et al., 2026). These safeguards act on the rewrite of the whole document, whose outcome is still judged by a few noisy rollouts. Skill^2 instead treats skill evolution as optimizing the policy that writes skills, and carries over three principles of reinforcement learning, namely a baseline for the return, credit assignment to individual design choices, and a replay buffer that keeps every evaluated skill informing later rounds.

3 Method

Section 3.1 states skill evolution as an optimization problem whose variable is a document and whose objective is task return. Section 3.2 maps this problem onto policy optimization, places existing methods in that correspondence, and identifies three mechanisms they leave out, namely a baseline, credit assignment, and a replay buffer. Section 3.3 presents Skill^2, which realizes all three through controlled comparisons of skills that differ in one design choice and keeps every evaluated skill in the context of the generator for reuse.

3.1 Formulation of the Skill Evolution Problem

A skill S is a document package, typically a `SKILL.md` with optional scripts and templates, loaded into the agent's context at inference time. With the weights θ frozen, the skill is the only free parameter of the agent policy.

The agent conditioned on S produces a trajectory $\tau \sim \pi_\theta(\cdot \mid S)$ on the task, and a verifier returns its score $R(\tau)$. The value of a skill is its expected return $J(S)$, and skill evolution seeks

$$S^* = \arg\max_S J(S), \qquad J(S) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|S)}\big[R(\tau)\big]. \tag{1}$$

J is observed only through rollouts: one run of S returns a trajectory τ with its score $r(S) = R(\tau)$.

To optimize Eq. 1, existing methods (Yang et al., 2026; Wang et al., 2026b; Shen et al., 2026) follow TextGrad (Yuksekgonul et al., 2024) and run textual gradient descent on S. An LLM $\mathcal{G}_{\text{grad}}(\cdot)$ reads one rollout of the current skill and returns natural-language feedback in place of a gradient, and a second $\mathcal{G}_{\text{patch}}(\cdot)$ call patches the skill with it:

$$S_{\text{new}} = \mathcal{G}_{\text{patch}}\Big(S, \frac{\partial J}{\partial S}\Big), \qquad \frac{\partial J}{\partial S} = \mathcal{G}_{\text{grad}}\big(S, \tau, R(\tau)\big). \tag{2}$$

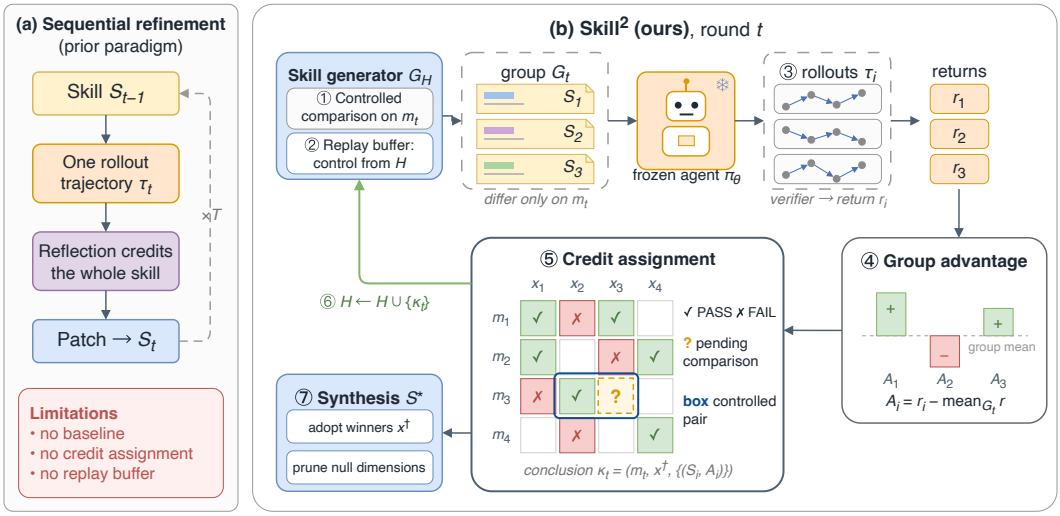


Figure 2: (a) Sequential refinement patches one skill from its own trajectory. (b) One round of Skill^2 compares skills that differ only on m_t and appends the conclusion κ_t to H.

Iterating this update round by round gives sequential refinement (Figure 2(a)), in which each rewrite rests on the single trajectory of the current skill.

On realistic multi-step tasks, a single trajectory of dozens of tool calls does not reveal which part of S caused its outcome, so optimizing S directly from such feedback lacks a reliable direction. Rather than treating the skill itself as the direct optimization target, we therefore treat skill evolution as a *policy* that generates skills. The policy is an LLM that writes S conditioned on a context H, which holds the evaluated skills together with their trajectories and returns; we write $S \sim \mathcal{G}_H(\cdot)$. The weights are frozen, so H is the sole parameter of the policy, and skill evolution becomes

$$H^* = \arg\max_H \; \mathbb{E}_{S\sim\mathcal{G}_H(\cdot)}\big[J(S)\big]. \tag{3}$$

3.2 Skill Evolution as Policy Optimization

In this section, we formalize skill evolution as policy optimization. We map each element of reinforcement learning onto skill evolution, then use this mapping to locate what existing methods leave out (Table 1).

Action. Section 3.1 optimizes the skill S as a monolithic document. To assign credit to individual design choices, we decompose it into choices on discrete *design dimensions*,

$$S = \big\langle x_m \big\rangle_{m\in\mathcal{M}}, \qquad x_m \in \mathcal{X}_m, \tag{4}$$

where $\mathcal{M}$ indexes the design dimensions of the skill, $\mathcal{X}_m$ is the discrete set of alternatives on dimension m, and $x_m \in \mathcal{X}_m$ is the choice S makes. An action is a single choice on one dimension.

Policy. The policy is the skill generator $\mathcal{G}_H$ (Eq. 3), an LLM that maps an evaluated skill to a new one,

$$S_{\text{new}} \sim \mathcal{G}_H(\cdot \mid S, \tau, r(S)), \tag{5}$$

where S_{new} differs from S on a subset of dimensions. The context H accumulates prior evaluations and is the sole parameter of the policy.

Environment. The environment $\mathcal{E}$ wraps the task instance, its workspace and tools, the frozen agent π_θ, and the verifier R. Given a skill S, it executes one rollout and returns a trajectory with its score,

$$\mathcal{E}(S) = \big(\tau,\ r(S)\big), \qquad \tau \sim \pi_\theta(\cdot \mid S), \quad r(S) = R(\tau). \tag{6}$$

RL estimator	Advantage term	Skill evolution counterpart	Representative method
REINFORCE	$r(\tau)$	reflect on one trajectory and rewrite the skill	Reflexion (Shinn et al., 2023)
Monte Carlo return	$\frac{1}{b}\sum_i r(\tau_i)$	run one skill on a batch of instances and merge the lessons	Trace2Skill (Ni et al., 2026)
actor-critic	$r(\tau) - V_\phi$	a co-evolved verifier scores the current skill	CoEvoSkills (Zhang et al., 2026b)
pairwise preference	$r(S^+) - r(S^-)$	contrast winning and losing variants in one patch	SkillEvolver (Zhang et al., 2026a)
evolution strategy	$\arg\max_i r_i$	sample several skills and keep the best	EvoSkill (Alzubi et al., 2026)
improvement check	$\mathbf{1}[\hat{J}(S') > \hat{J}(S)]$	accept a rewrite only if it scores higher	SkillAdaptor (Yu et al., 2026)

Table 1: RL advantage estimators and their skill evolution counterparts.

Return. The expected return $J(S) = \mathbb{E}_{\tau\sim\pi_\theta(\cdot|S)}[R(\tau)]$ (Eq. 1) is observed only through rollouts. Each rollout is a full agent execution in a containerized environment, so we observe one sample per skill,

$$\hat{J}(S) = r(S) = R(\tau), \qquad \tau \sim \pi_\theta(\cdot \mid S). \tag{7}$$

The context keeps τ alongside $r(S)$, since the return alone does not identify the behavior that produced it.

Advantage. The advantage of an action measures how much better it is than the skill it is applied to. Starting from an evaluated skill S, the advantage of replacing the choice on dimension m with alternative x is

$$A(m{\leftarrow}x \mid S) = J\big(S[m{\leftarrow}x]\big) - J(S), \tag{8}$$

where $S[m{\leftarrow}x]$ is the skill that agrees with S on every dimension except m, which takes x. Table 1 classifies existing skill evolution methods by how they estimate this quantity.

Table 1 reads existing updates against Eq. 8. Reflection on one trajectory and Monte Carlo returns score a skill with no reference, so the update has no baseline. Critics, pairwise contrasts, and improvement checks do compare two skills, but the pair differs on many dimensions at once, so the gap is credited to the whole document and not to a choice, and there is no credit assignment. Methods that keep memory across rounds (Yang et al., 2026; Wang et al., 2026b) store rejected edits or recurring diagnoses. They keep what was concluded about past skills but discard the evaluated skills and their trajectories, so no later round can reuse them as controls, and there is no replay buffer. Skill^2 constructs entries that estimate Eq. 8 directly.

3.3 SKILL^2

Skill^2 closes the three gaps of Section 3.2 by growing the context H one controlled comparison at a time. Figure 2(b) shows one round, and Algorithm 1 gives the full loop.

Controlled comparison. In round t the skill generator reads H and picks a design dimension m_t that H has not resolved, such as whether a vulnerability check should rely on an offline scanner or on a live advisory database. It then builds a group whose members differ only on m_t,

$$G_t = \big\{\, S_{\mathrm{ctrl}}[m_t{\leftarrow}x] \;:\; x \in \mathcal{X}_t \,\big\}, \tag{9}$$

where $\mathcal{X}_t \subseteq \mathcal{X}_{m_t}$ holds the compared choices and the control S_{ctrl} fixes all other dimensions. The generator first targets steps that every correct solution must perform but that H has never varied, then behaviors that passing trajectories show but no skill prescribes. The first round has no control, so the generator samples two or three skills with independent overall strategies.

Replay buffer. The return of a skill depends only on the skill and the frozen agent, so every evaluated skill in H remains valid evidence together with its trajectory and return. When H already holds a skill on one side of the comparison, that skill serves as S_{ctrl} without being executed again,

Algorithm 1 Skill^2

Require: task with seed instance; rollout budget
Ensure: final skill S^*
$H \leftarrow \emptyset$
while budget remains **do**
  $m \leftarrow$ GEN.CHOOSEDIMENSION(H)
  $G \leftarrow$ GEN.BUILDGROUP(H, m) ▷ Eq. 9, reuses controls in H
  for each new $S_i \in G$ **do**
    $(\tau_i, r_i) \leftarrow \mathcal{E}(S_i)$; append to H
  end for
  $\kappa \leftarrow$ GEN.COMPARE(G, H); append to H ▷ Eq. 10
end while
return GEN.SYNTHESIZE(H) ▷ Eq. 11

and the round costs a single rollout. Otherwise the generator writes two new skills. Sequential refinement cannot reuse experience in this way, because its feedback in Eq. 2 describes a skill that no longer exists once it is patched.

Group advantage. The frozen agent executes each new member once on the seed instance, and the verifier scores the trajectory. The generator sees the score and the trajectory but not test names or failure messages. Since the members share every choice except the one on m_t, the control serves as the baseline. For a member $S_i = S_{\text{ctrl}}[m_t \leftarrow x_i]$, the score difference $A_i = r(S_i) - r(S_{\text{ctrl}})$ is a one-rollout estimate of $A(m_t \leftarrow x_i \mid S_{\text{ctrl}})$ in Eq. 8. When all members tie, the generator compares trajectories instead and favors the member that reaches the result with fewer retries and fewer deviations from its skill.

Credit assignment. The generator credits the difference in outcomes to m_t only where the trajectories diverge at a step the skill prescribed. A success the agent reached against or beyond its skill is not credited, since the skill did not cause it. The round ends with a conclusion

$$\kappa_t = \big(m_t,\ x^\dagger,\ \{(S_i, A_i)\}_{S_i \in G_t}\big), \tag{10}$$

which records the dimension, the supported choice $x^\dagger$, and the advantage of each member. When changing m_t alters neither the score nor the behavior, the conclusion is null and marks the dimension as irrelevant to the task. Appending κ_t to H updates the policy and shapes the question of the next round.

Synthesis. Once the budget is spent, the generator compiles H into the final skill without further rollouts. It starts from the best evaluated skill $\hat{S}$, rewrites any text that the passing trajectory of $\hat{S}$ contradicts, and applies every resolved conclusion,

$$S^* = \hat{S}\big[m \leftarrow x^\dagger_m\big]_{m \in \mathcal{R}}, \tag{11}$$

where $\mathcal{R}$ collects the dimensions with a non-null conclusion. Text on null dimensions is removed, and any added sentence without support from a passing trajectory is reverted. S^* may never have been evaluated as a whole, yet each of its choices has won a controlled comparison.

4 EXPERIMENTS

4.1 SETUP

We evaluate on SkillLearnBench (Zhong et al., 2026), which contains 20 real-world tasks across software engineering, information retrieval, productivity tools, data and analysis, content generation, and utilities. Each task has two to six instances, 100 in total, and an automated verifier grades each instance in an isolated Docker container. Every method writes one skill per task from the instruction of the seed instance, the first instance of the task, and this skill is then evaluated on every instance of the task. We run three models, DeepSeek-V4-Flash, GPT-5.6-Luna, and Gemini-3.8-Flash. Each model serves as both the skill generator and the frozen agent, which runs in Claude Code (Anthropic,

Table 2: Skill quality (0–100) and task accuracy (%) on SkillLearnBench. **Bold** marks the best automated method, and Human-Authored is an expert reference.

		Skill quality			Accuracy (%)	
Model	**Method**	**Completeness**	**Determinism**	**Consistency**	**All**	**Held-out**
DeepSeek-V4-Flash	No Skill	–	–	–	29.7	28.5
	One-Shot	37.8	39.9	71.8	29.5	26.2
	Self-Feedback	33.1	36.2	65.4	30.8	33.0
	Skill^2 (ours)	**65.8**	**61.5**	**87.7**	**40.8**	**40.0**
	Human-Authored (ref.)	38.6	39.6	74.0	58.5	56.8
GPT-5.6-Luna	No Skill	–	–	–	46.0	43.0
	One-Shot	45.2	44.9	80.3	39.2	40.8
	Self-Feedback	35.2	37.0	68.6	38.5	37.0
	Skill^2 (ours)	**47.1**	**46.1**	**87.4**	**46.7**	**48.3**
	Human-Authored (ref.)	38.6	39.6	74.0	66.7	67.0
Gemini-3.8-Flash	No Skill	–	–	–	52.2	54.8
	One-Shot	33.5	37.3	**72.1**	49.7	52.2
	Self-Feedback	33.0	37.3	70.6	53.5	51.2
	Skill^2 (ours)	**41.3**	**44.9**	70.5	**68.5**	**68.2**
	Human-Authored (ref.)	38.6	39.6	74.0	74.3	73.8

2025a), Codex CLI (OpenAI, 2025), and Gemini CLI, respectively. Skill^2 has a budget of 10 rollouts on the seed instance, and the generator decides from the context how many rounds to run and how many skills each group contains.

Baselines. All baselines follow the implementations released with SkillLearnBench, and we reuse their prompts unchanged. NO SKILL runs the frozen agent without any skill. ONE-SHOT asks the generator to analyze the seed instruction and write one to five modular skills in a single pass, without executing them. SELF-FEEDBACK is the sequential refinement baseline of SkillLearnBench. In the first round the agent writes skills and attempts the seed instance. In the second round it reviews its own attempt, revises the skills, and attempts the instance again. It receives no verifier result, and we evaluate the revised skills. We keep the two rounds of the original configuration because further rounds hurt this baseline. When Zhong et al. (2026) extend SELF-FEEDBACK to four rounds, accuracy peaks after the first revision and then falls sharply (Figure 1(a)). Two rounds is therefore its most favorable setting. HUMAN-AUTHORED evaluates the expert-written skills released with SkillLearnBench. All methods share the same instances, frozen agent, and verifier.

Metrics. We report skill quality and task accuracy. Skill quality follows the executability evaluation of SkillLearnBench. A GPT-5-mini judge scores each skill on a 1 to 5 scale once for every instance of its task, and we rescale the scores to 0 to 100 and average them within each task and then across tasks. *Completeness* measures whether the skill specifies every step, tool, and dependency that execution requires. *Determinism* measures whether its instructions and decision rules are precise enough to follow without guessing. *Consistency* measures whether it is free of logical and technical contradictions. Task accuracy is the verifier pass rate, averaged within each task and then across the 20 tasks. *All* covers the 100 instances, and *Held-out* excludes the 20 seed instances, so it measures transfer to the 80 instances that no method sees during generation.

4.2 MAIN RESULTS

Table 2 reports the results. Skill^2 attains the highest accuracy among automated methods for all three models, both on all instances and on held-out instances. Relative to NO SKILL, it gains 11.1, 0.7, and 16.3 points on all instances for DeepSeek-V4-Flash, GPT-5.6-Luna, and Gemini-3.8-Flash, and 11.5, 5.3, and 13.4 points on held-out instances. Excluding the seed instance lowers its accuracy by at most 0.8 points, so the gain does not come from fitting the seed. HUMAN-AUTHORED skills remain strongest at 58.5%, 66.7%, and 74.3%, so Skill^2 narrows the gap to expert skills.

The two SkillLearnBench baselines do not improve on the bare agent. ONE-SHOT falls below NO SKILL for every model, by 0.2 to 6.8 points, consistent with the difficulty of writing a useful skill from one instruction. SELF-FEEDBACK stays within 1.3 points of NO SKILL for DeepSeek-V4-Flash and Gemini-3.8-Flash and falls 7.5 points below it for GPT-5.6-Luna. This matches the diagnosis of Section 1. A rewrite made from one unanchored attempt cannot tell which design caused the outcome, so the second round adds little.

On skill quality, Skill^2 scores highest in completeness and determinism for every model, and above HUMAN-AUTHORED skills on both. Its consistency is the highest for DeepSeek-V4-Flash and GPT-5.6-Luna and within 1.6 points of the best for Gemini-3.8-Flash. The margins concentrate in completeness and determinism, which suggests that the accuracy gain comes from supplying more of the information and decision rules that execution needs rather than from removing contradictions.

4.3 ABLATION STUDY

We isolate three choices in the skill generator. W/O CONTROLLED COMPARISON retains prior skills and scores but lets the skill generator propose new candidates freely, without requiring a one-dimension contrast or writing a dimension-level conclusion to the context. W/O CONTROL REUSE still asks one dimension-level question per round but constructs each comparison from newly generated candidates rather than reusing a historical candidate as a control. W/O SYNTHESIS reuses the full context but exports the highest-scoring historical candidate instead of synthesizing the winning dimension values into a final skill. We run these variants with the same rollout budget on five tasks from the Productivity Tools and Information Retrieval categories. The GPT-5-mini judge scores completeness, determinism, consistency, and usability, rescaled to 0 to 100 and averaged as in Table 2.

Table 3: Skill-quality ablations on five tasks with GPT-5.6-Luna (0–100).

Variant	Comp.	Det.	Cons.	Usab.
Skill^2 (full)	**73.2**	**73.7**	**96.3**	**80.8**
w/o Controlled Comp.	72.2	63.3	96.2	79.2
w/o Control Reuse	66.5	62.0	92.2	76.0
w/o Synthesis	70.2	63.0	92.5	77.3

The full method receives the highest mean judgment on all four dimensions. The clearest separation is determinism. Removing controlled comparison lowers it from 73.7 to 63.3, removing control reuse lowers it to 62.0, and removing synthesis lowers it to 63.0. Completeness and consistency change little without controlled comparison, whereas forbidding control reuse reduces all four dimensions, including completeness from 73.2 to 66.5.

4.4 CASE STUDY

We trace the complete DeepSeek-V4-Flash search on earthquake-plate-calculation. The agent must find the 2024 earthquake inside the Pacific plate that lies farthest from the plate boundary and report its identity and that distance in kilometers. Figure 3 summarizes the search on the left and plots, on the right, how many of the eight verifier tests each evaluated skill passes on the seed instance. The final skill passes four of six evaluation instances. HUMAN-AUTHORED and SELF-FEEDBACK pass one, and ONE-SHOT and NO SKILL pass none.

Equal scores carry no advantage. Round 1 samples three strategies, namely a GeoPandas pipeline, explicit point-to-segment geometry, and the plate polygon's own ring as the boundary. All three pass seven of eight tests, so every group advantage is zero, and the score cannot rank them. The generator is never told which test fails, but from the trajectories it observes that all three report the same epicenter and differ only in the distance they compute, which leaves the distance definition as the open dimension.

Controlled pairs resolve one dimension at a time. Round 2 writes two skills that share every step and differ only in the projection used for distance. EPSG:4087 passes all eight tests and ESRI:53032 passes seven, so the advantage is credited to the projection. Round 3 consolidates the winner into a single skill, which again passes all eight tests and serves as the control for later rounds. Round 4 varies whether the agent re-derives the reported value in a final check or carries it forward from the analysis. Both pass, and the null conclusion keeps the check out of the skill. Round 5 returns to

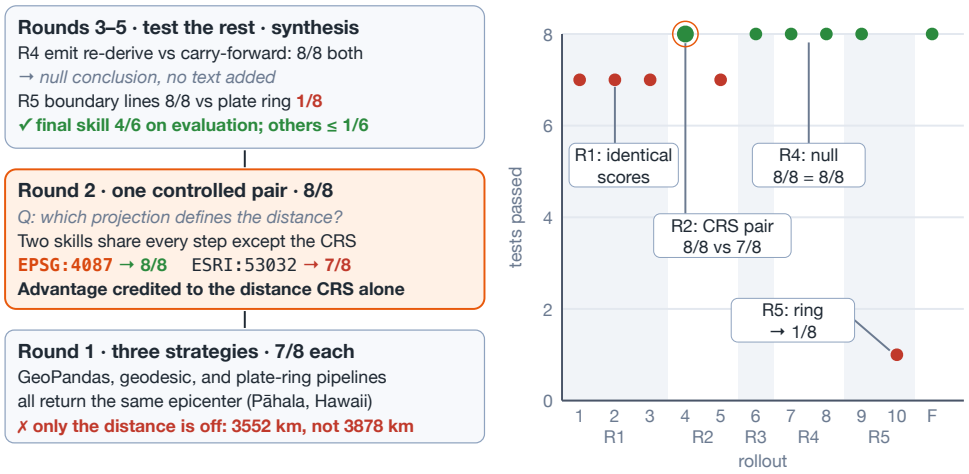


Figure 3: Skill^2 search on earthquake-plate-calculation with DeepSeek-V4-Flash. Left, the three search phases. Right, seed-instance tests passed by each evaluated skill, grouped by round.

the boundary, which round 1 had changed together with the strategy. With the rest of the pipeline fixed, the boundary-line layer passes and the plate ring passes one test, because seam segments at the antimeridian change which earthquake wins. The low score belongs to a tested alternative, not to the carried-forward skill. A plausible reading of the task is thereby ruled out rather than left in the skill as an alternative.

Synthesis keeps only supported choices. Synthesis anchors on r5v1, which already carries the winning value of every resolved dimension. Its audit finds that the passing agent followed each prescribed step, so no sentence is rewritten, and the null conclusion from round 4 adds nothing. HUMAN-AUTHORED also prescribes EPSG:4087 yet passes one evaluation instance. The resulting skill thus differs from a generic GeoPandas recipe less in any single choice than in the fact that each of its choices, including what it omits, is backed by a comparison on the seed instance.

5 CONCLUSION

We presented Skill^2, which casts skill evolution as optimizing the policy that writes skills. Under this view, the baseline, credit assignment, and replay buffer of policy-gradient methods have natural-language counterparts that sequential refinement lacks. Controlled comparisons credit score differences to single design choices, and the context keeps every evaluated skill as a reusable control. On SkillLearnBench, Skill^2 outperforms ONE-SHOT and SELF-FEEDBACK for each of three LLMs, and removing any of its components lowers judged skill quality.

Limitations. The correspondence to policy optimization is an analogy, not a formalism. Advantages and attributions are computed in natural language, and we claim no convergence guarantee. Skill^2 spends multiple costly agent rollouts per task, which may be impractical for large task sets or expensive environments. Its questions, dimensions, and controls are proposed by an LLM and may still be confounded, and a single controlled comparison provides defeasible evidence, not causal proof. Its gain also depends on the capability of the underlying model and is small for GPT-5.6-Luna.

AI USE STATEMENT

We used generative AI tools to polish the writing of the manuscript, to search for related work, and to assist in writing experiment code. The authors reviewed and verified all AI-assisted content and take full responsibility for this work.

REFERENCES

Salaheddin Alzubi, N. Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated Skill Discovery for Multi-Agent Systems. *arXiv preprint arXiv:2603.02766*, 2026. URL https://arxiv.org/pdf/2603.02766.

Anthropic. Claude Code: An Agentic Coding Tool. https://github.com/anthropics/claude-code, 2025a.

Anthropic. Equipping Agents for the Real World with Agent Skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, 2025b. Accessed: 2026-03-30.

Anthropic. Skill-Creator: Official Anthropic Agent Skill for Authoring Skills. https://github.com/anthropics/skills/tree/main/skills/skill-creator, 2025c. Accessed: 2026-03-30.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? *arXiv preprint arXiv:2603.15401*, 2026. doi: 10.48550/arXiv.2603.15401. URL https://arxiv.org/pdf/2603.15401.

Sihang Jiang, Li-Ping Ma, Z. Hong, Keyi Wang, Zhiyu Lu, Shi-Song Chen, Jinghao Zhang, Tian Pan, Weijia Zhou, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A Benchmark for Evaluating Self-Evolving Agents Beyond Episodic Assessment. *arXiv preprint arXiv:2604.08988*, 2026. doi: 10.48550/arXiv.2604.08988.

Xiangyi Li et al. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670*, 2026a. doi: 10.48550/arXiv.2602.12670. URL https://arxiv.org/pdf/2602.12670.

Yu Li, Rui Miao, Zhengling Qi, and Tian Lan. ARISE: Agent Reasoning with Intrinsic Skill Evolution in Hierarchical Reinforcement Learning. *arXiv preprint arXiv:2603.16060*, 2026b. URL https://arxiv.org/pdf/2603.16060.

Xing-Yan Liu, Xi-Yue Luo, Lin-Yu Li, Gang Huang, Jian-Feng Liu, and Hongli Qiao. SkillForge: Forging Domain-Specific, Self-Evolving Agent Skills in Cloud Technical Support. *arXiv preprint arXiv:2604.08618*, 2026a. doi: 10.1145/3805712.3808466. URL https://doi.org/10.1145/3805712.3808466.

Yujian Liu, Jiabao Ji, Li An, T. Jaakkola, Yang Zhang, and Shiyu Chang. How Well Do Agentic Skills Work in the Wild: Benchmarking LLM Skill Usage in Realistic Settings. *arXiv preprint arXiv:2604.04323*, 2026b. doi: 10.48550/arXiv.2604.04323.

Zhengxi Lu et al. SKILL0: In-Context Agentic Reinforcement Learning for Skill Internalization. *arXiv preprint arXiv:2604.02268*, 2026. URL https://arxiv.org/pdf/2604.02268.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. SkillClaw: Let Skills Evolve Collectively with Agentic Evolver. *arXiv preprint arXiv:2604.08377*, 2026. URL https://arxiv.org/pdf/2604.08377v1.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-Pro: Learning Reusable Skills from Experience via Non-Parametric PPO for LLM Agents. *arXiv preprint arXiv:2602.01869*, 2026. URL https://arxiv.org/pdf/2602.01869.

Jingwei Ni et al. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills. *arXiv preprint arXiv:2603.25158*, 2026. URL https://arxiv.org/pdf/2603.25158.

OpenAI. Codex CLI: Lightweight Coding Agent that Runs in Your Terminal. https://github.com/openai/codex, 2025.

Siru Ouyang et al. SkillOS: Learning Skill Curation for Self-Evolving Agents. *arXiv preprint arXiv:2605.06614*, 2026. URL https://arxiv.org/pdf/2605.06614v1.

Yifei Shen, Bo Li, and Xinjie Zhang. SkillOpt-Lite: Better and Faster Agent Self-evolution via One Line of Vibe. *arXiv preprint arXiv:2607.03451*, 2026.

Noah Shinn, Federico Cassano, Beck Labash, A. Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.52202/075280-0377. URL https://arxiv.org/pdf/2303.11366.

Yu Tian, Jiawei Chen, Lifang Zheng, Ming Tao, Xinyi Zeng, Zhaoxia Yin, Hang Su, and Xian Sun. Skills-Coach: A Self-Evolving Skill Optimizer via Training-Free GRPO. *arXiv preprint arXiv:2604.27488*, 2026. URL https://arxiv.org/pdf/2604.27488.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, and Dongbin Zhao. Dynamic Dual-Granularity Skill Bank for Agentic RL. *arXiv preprint arXiv:2603.28716*, 2026. URL https://arxiv.org/pdf/2603.28716.

Chenxi Wang et al. SkillX: Automatically Constructing Skill Knowledge Bases for Agents. *arXiv preprint arXiv:2604.04804*, 2026a. URL https://arxiv.org/pdf/2604.04804.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. SkillGrad: Optimizing Agent Skills Like Gradient Descent. *arXiv preprint arXiv:2605.27760*, 2026b. doi: 10.48550/arXiv.2605.27760.

Peng Xia et al. SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforcement Learning. *arXiv preprint arXiv:2602.08234*, 2026. doi: 10.48550/arXiv.2602.08234. URL https://arxiv.org/pdf/2602.08234v1.

Renjun Xu and Yang Yan. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. *arXiv preprint arXiv:2602.12430*, 2026. doi: 10.48550/arXiv.2602.12430. URL https://arxiv.org/pdf/2602.12430.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue-Ting Yang, and Chong Luo. SkillOpt: Executive Strategy for Self-Evolving Agent Skills. *arXiv preprint arXiv:2605.23904*, 2026. doi: 10.48550/arXiv.2605.23904.

Zhuoyun Yu, Xinghong Xie, Wuguannan Yao, Chenxi Wang, Lei Liang, Xiang Qi, and Shumin Deng. SkillAdaptor: Self-Adapting Skills for LLM Agents from Trajectories. *arXiv preprint arXiv:2606.01311*, 2026. doi: 10.48550/arXiv.2606.01311. URL https://arxiv.org/pdf/2606.01311.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "Differentiation" via Text. *arXiv preprint arXiv:2406.07496*, 2024. doi: 10.48550/arXiv.2406.07496.

Ge Zhang, Erle Zhu, Jinfeng Zhou, Caiyan Jia, and Hongning Wang. SkillEvolver: Skill Learning as a Meta-Skill. *arXiv preprint arXiv:2605.10500*, 2026a. doi: 10.48550/arXiv.2605.10500. URL https://arxiv.org/pdf/2605.10500.

Hanrong Zhang et al. CoEvoSkills: Self-Evolving Agent Skills via Co-Evolutionary Verification. *arXiv preprint arXiv:2604.01687*, 2026b. URL https://arxiv.org/pdf/2604.01687.

Boyuan Zheng et al. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *arXiv preprint arXiv:2504.07079*, 2025. doi: 10.48550/arXiv.2504.07079. URL https://arxiv.org/pdf/2504.07079.

Shan Zhong et al. SkillLearnBench: Benchmarking Continual Learning Methods for Agent Skill Generation on Real-World Tasks. *arXiv preprint arXiv:2604.20087*, 2026. URL https://arxiv.org/pdf/2604.20087.

Huichi Zhou et al. Memento-Skills: Let Agents Design Agents. *arXiv preprint arXiv:2603.18743*, 2026. URL https://arxiv.org/pdf/2603.18743v1.

APPENDIX

A SkillLearnBench Split **12**

B Skill-Quality Judge Prompt **13**

C Skill Generator Prompts **13**

C.1 Round 1 13

C.2 Later Rounds 14

C.3 Synthesis 15

C.4 Shared Rules 15

A SKILLLEARNBENCH SPLIT

SkillLearnBench (Zhong et al., 2026) organizes its 20 tasks into 6 categories and 15 sub-domains (Table 4). Each task defines a workflow, and its instances vary the inputs, entities, and constraints of that workflow, so a skill written for a task has to carry the procedure rather than the answer to one instance. Each instance ships a Docker image, a natural-language instruction, and a verifier written as a pytest suite, and it counts as solved only when every test passes.

The first instance of each task is its seed instance. Every method receives only the seed instruction when it writes the skill for a task, and Skill^2 executes its candidate skills on the seed instance alone. The remaining 80 instances form the held-out split and are never seen during generation. The *All* column of Table 2 averages over the 100 instances and the *Held-out* column over these 80. Both are macro averages, computed first within each task and then across the 20 tasks, so a task with two instances weighs as much as a task with six.

Table 4: Task taxonomy and split of SkillLearnBench.

Category	Sub-domain	Task	#Inst.	Seed	Held-out
Software Engineering	Code Generation	python-scala-translation	2	1	1
		nlp-paper-reproduction	3	1	2
	Debug & Analysis	dependency-vulnerability-check	5	1	4
	Version Control	github-repo-analytics	5	1	4
	Infrastructure	fix-security-bug	3	1	2
Information Retrieval	Web Search	enterprise-information-search	6	1	5
		travel-planning	5	1	4
Productivity Tools	Team Communication	schedule-planning	5	1	4
	Document Systems	offer-letter-generator	6	1	5
		court-form-filling	6	1	5
Data & Analytics	Data Processing	earthquake-plate-calculation	6	1	5
		financial-analysis	6	1	5
	Math & Calculation	weighted-gdp-calculation	6	1	5
		dbscan-parameter-tuning	5	1	4
	Data Visualization	stock-data-visualization	5	1	4
Content & Creative	Image Generation	anthropic-poster-design	5	1	4
	Text Generation	chinese-poem-generator	5	1	4
	Audio & Video	video-object-counting	5	1	4
Utilities & Other	Local File Control	organize-messy-files	6	1	5
	Command Execution	temperature-simulation	5	1	4
Total: 6 categories, 15 sub-domains, 20 tasks			100	20	80

B SKILL-QUALITY JUDGE PROMPT

Skill quality in Table 2 follows the executability evaluation of SkillLearnBench. The judge receives the prompt below once for every instance of a task, with the instance instruction and the full skill package filled in, and returns an integer score from 1 to 5 on each dimension. We map a score s to $25(s-1)$.

```
# Task Background
You are an **expert evaluator of agent skills used in autonomous
    AI systems**.
Your goal is to judge whether the provided skill specification
   could be **reliably executed by an agent in practice**.
You must evaluate the skill as if an agent **blindly follows the
    instructions step-by-step**.
If execution would fail, stall, or require missing information,
   this must strongly reduce the score.
# Input
## Task Description
<task instruction>
## AI-generated agent skill
<skill package>
# Critical Instruction
Before scoring, you must perform a **mental execution simulation
   **.
Imagine an agent attempting to execute the skill exactly as
   written.
During this simulation check:
- Are all required steps executable?
- Are required inputs available?
- Are tools clearly defined?
- Are resources accessible?
- Would any step fail or become undefined?
If execution would **fail or stall**, the score must be low.
Do NOT judge based on writing quality or style.
Judge based on **actual executability**.
# Evaluation Dimensions
## 1. Completeness
Does the skill contain **all information required for execution
   **?
Check for missing:
- execution steps
- required parameters
- tool definitions
- environment requirements
- file paths or resources
- expected outputs
If the agent cannot execute the skill due to missing information,
    the score must be <=2.
## 2. Determinism
Are the instructions **precise enough that the agent knows
   exactly what to do**?
Penalize:
- vague instructions
- missing decision rules
- unclear branching conditions
- underspecified parameters
- instructions requiring interpretation
If the agent would need to guess what to do next, the score must
    be <=2.
## 3. Consistency
Check for **logical and technical contradictions**.
Examples:
- conflicting steps
- inconsistent variable names
- undefined variables
- references to non-existent tools
- incompatible workflow steps
If contradictions would cause execution errors, the score must
   be <=2.
DO NOT judge based on writing quality or style.
## 4. Usability
Can the skill be reused across different tasks?
Penalize:
- hardcoded task-specific artifacts
- temporary file paths
- instance-specific data embedded in the skill
- dependencies on one specific environment
Reusable designs should score higher.
# Failure Detection (Important)
During evaluation explicitly check:
1. Missing execution information
2. Undefined variables or tools
3. Steps that cannot actually run
4. Resources that are not provided
5. Ambiguous instructions requiring interpretation
6. Hardcoded task-specific values
If any of these issues would break execution, reduce the score.
# Scoring Scale
Use the following strict interpretation:
**1 -- Not executable**
Execution would definitely fail or stall due to missing
   information or undefined resources.
**2 -- Severely flawed**
Execution contains errors that would probably cause failure or
   stall.
**3 -- Partially executable**
Some parts are runnable but important information is missing or
   ambiguous.
**4 -- Mostly executable**
Minor issues exist but an agent could probably complete the task
    with small assumptions.
**5 -- Fully executable**
Instructions are clear, complete, deterministic, and reusable.
Scores of **5 should be rare** and only given if execution would
    succeed without assumptions.
# Thinking Instruction
Follow this reasoning order:
1. Simulate the execution of the skill step-by-step.
2. Identify where execution would fail, stall, or require
   guessing.
3. Evaluate each dimension independently.
4. Assign scores conservatively.
Do not give high scores unless the skill is clearly executable.
# Output Format
Return the evaluation result strictly in the following JSON
   format:
{
  "completeness_reason": "...",
  "completeness_score": 1-5,
  "determinism_reason": "...",
  "determinism_score": 1-5,
  "consistency_reason": "...",
  "consistency_score": 1-5,
  "usability_reason": "...",
  "usability_score": 1-5
}
```

C SKILL GENERATOR PROMPTS

We reproduce the three prompts of the Skill^2 generator verbatim. Placeholders in angle brackets are filled at run time, and the shared rules they reference follow the prompts. In the main experiments the visibility rule is appended to every prompt.

C.1 ROUND 1

```
You are the Skill Generator in an AI agent pipeline.
Total rollout budget: B = {B}. This is Round 1 -- there is no
   execution history,
so you cannot reference any result. The only question you can
   ask at this point
is the most macroscopic one: "which overall strategy best fits
   this task?"
Answering that question requires putting several INDEPENDENT
   strategies side by
```

```
side. So generate 2 to 3 skill PACKAGES, each embodying a
   DIFFERENT overall
strategy. They are not refinements of one another -- they are
   distinct
hypotheses about how to approach the task.
A skill is a DIRECTORY (a structured package), not a single flat
    file. At
minimum it contains SKILL.md. You MAY add supporting files that
   help the
executing agent -- scripts, templates, worked examples, a
   detailed procedure.
Use progressive disclosure in SKILL.md: a short overview and
   core workflow
first, details and edge cases after.
<META SPEC: declares question, dimension, value, and group in
   meta.json>
For Round 1 the comparison group is empty:
  {"question": "which overall strategy best fits this task",
    "dimension": "overall_strategy", "value": "<this strategy's
   name>",
    "group": []}
Write each package under its own directory:
  environment/skills/variant_1/SKILL.md  + meta.json [+ extra
   files]
  environment/skills/variant_2/...
  environment/skills/variant_3/...        (if a third strategy
   is worth testing)
CRITICAL OUTPUT CONTRACT: the top-level package directory names
   MUST be exactly
`variant_1`, `variant_2`, and optionally `variant_3`. Do NOT use
    descriptive
directory names such as `eager_sequential`, `constraint_aware`,
   or
`optimization_based`; put descriptive names only in SKILL.md
   frontmatter and
meta.json `value`. The runner ignores every package outside `
   variant_*`.
Each SKILL.md must begin with a YAML frontmatter block:
  ---
  name: <skill-name>
  description: <one-sentence description>
  ---
<GENERALIZATION RULE>
Task description:
<seed instruction>
```

C.2 Later Rounds

```
\
You are the Skill Generator in an AI agent pipeline.
You are NOT "improving the previous skill". Each round you run
   the cheapest
experiment that most reduces uncertainty about one question:
   what must a skill
make the actor DO so that it solves this task correctly -- and
   would still solve
a different instance of it?
Remaining rollout budget: {remaining}. Every new skill you
   generate this round
costs one rollout. Historical trials cost nothing -- they are
   free comparison
points you reuse.
All prior experiments live under /root/history/<id>/:
  result.json    -- the global map for this trial: question/
   dimension/value/group
                    plus its outcome (status/score/passed_tests/
   failed_tests)
  skill/SKILL.md [+ package files]  -- the skill that was tried
  verifier.txt   -- full verifier output
  trajectory.txt -- the actor's full execution trace
Complete these steps in order:
STEP 1 -- Reconstruct what actually happened, not just what was
   scored.
  Read every result.json to map which dimensions and values have
    been tested,
  and which pairs of trials form CLEAN comparisons (differ on
   exactly one
  dimension) versus CONFOUNDED ones. Then read the trajectories
   of at least the
  best and the worst trial, asking for each: which concrete sub-
   steps did the
  actor perform to earn its score -- and did it perform them
   BECAUSE the skill
  said so, DESPITE what the skill said, or on its own initiative
   ? A skill can
  only reproduce a success it actually caused; whatever the
   actor supplied on
  its own is design that has never been tested.
STEP 2 -- Choose the one question worth this round's rollout(s).
  Candidate questions come from three sources, in priority order:

  (a) UNCOVERED CRITICAL SUB-STEP. From the task description and
    the verifier's
     test names, list the sub-steps any correct solution must
   get right --
     extracting every stated constraint, computing every
   required quantity,
     checking every requirement against its bound before
   finalizing. If no
     comparison has ever varied HOW the skill handles one of
   these, that gap
     outranks everything else: an untested critical step is
   where the next
     failure hides.
  (b) TEXT-BEHAVIOR DIVERGENCE. Wherever STEP 1 found a passing
   actor deviating
     from its skill or silently supplying a step its skill
   never mentioned,
     whether the skill should prescribe that behavior
   explicitly is a
     testable dimension.
  (c) UNRESOLVED MATRIX CELL. A confounded comparison to make
   clean, or a
     tested dimension with an informative untested value.
  Phrase the choice as a DIMENSION-LEVEL question -- should a
   skill have X, or
  use approach A vs B -- never as "fix a specific problem in
   r7v1". Meta-level
  dimensions (tool choice, verbosity, formatting, workflow style
   ) are worth a
  rollout only when sources (a) and (b) are exhausted. When
   every trial passes
  with the same score, pass/fail stops discriminating and the
   outcome signal
  becomes trajectory evidence -- deviations, self-corrections,
   retries,
  instance-specific dependence -- but the priority order above
   still applies.
  If (a), (b) and (c) are all genuinely exhausted, the task is
   insensitive to
  design and the final skill should favour the simplest passing
   version.
STEP 3 -- Generate the MINIMUM skills needed to answer it.
  - If one side of the comparison already exists in history ->
   generate only the
     OTHER side: take that historical trial's entire design and
   change ONLY your
     chosen dimension so it aligns into a clean controlled pair
   -> g = 1.
  - If neither side exists yet -> generate two skills sharing
   the same base, with
     that one dimension set to two different values -> g = 2.
  g is determined by the question and what history already
   covers -- do NOT
  collapse to g = 1 just to save effort when the comparison
   genuinely needs two
  new points. But generate AT MOST {remaining} new skill(s) this
    round: if the
  cleanest question needs two new points and only one rollout
   remains, pick a
  different question that one rollout can answer. A skill beyond
    the remaining
  budget is silently dropped, which would leave its comparison
   half-built.
  Each new skill declares in meta.json its question/dimension/
   value/group, where
  group lists the historical trial id(s) it forms a controlled
   comparison with.
A skill is a structured DIRECTORY (SKILL.md + optional scripts/
   templates/
examples/procedure), with progressive disclosure.
<META SPEC: declares question, dimension, value, and group in
   meta.json>
Write each package under:
  environment/skills/variant_1/SKILL.md  + meta.json [+ extra
   files]
  environment/skills/variant_2/...       (only if the question
   needs a second new point)
CRITICAL OUTPUT CONTRACT: the top-level package directory names
   MUST be exactly
`variant_1` and, when needed, `variant_2`. Do NOT use
   descriptive directory
names such as `grep_guided`, `concise_workflow`, or `
   constraint_solver`; put
descriptive names only in SKILL.md frontmatter and meta.json `
   value`. The runner
ignores every package outside `variant_*`.
Each SKILL.md must begin with a YAML frontmatter block:
  ---
  name: <skill-name>
  description: <one-sentence description>
```

```
  ---
<GENERALIZATION RULE>
<HONESTY RULE>
Task description:
<seed instruction>
```

C.3 SYNTHESIS

```
You are the Skill Generator in an AI agent pipeline.
This is the FINAL round. It does NOT consume rollout budget --
   which cuts both
ways: the skill you write will be deployed WITHOUT any further
   verification.
A sentence no passing rollout ever executed is an untested
   hypothesis, and this
round cannot test it. So you do not write a new skill; you
   deliver the most
verified one, minimally edited under evidence.
All prior experiments are under /root/history/<id>/ (result.json,
    skill/,
verifier.txt, trajectory.txt).
STEP 1 -- Anchor. Pick the highest-scoring trial (break ties
   toward the
  trajectory with the fewest deviations, retries, and instance-
    specific
  dependencies). Copy its ENTIRE package -- SKILL.md and every
    supporting
  file -- as the final skill's starting point. Where no evidence
    bears on a
  part of the anchor, that part ships unchanged.
STEP 2 -- Audit the anchor against its own trajectory. Read the
   anchor trial's
  trajectory.txt IN FULL -- the decisive judgment calls often
    sit in the middle
  or end of the trace, so never skim only its beginning.
    Wherever the passing
  actor's actual behavior contradicts the anchor's text -- the
    text says do Y,
  the actor did X and the trial still passed -- the behavior is
    what was
  verified, not the sentence. Rewrite that sentence to prescribe
    what the
  passing actor actually did. Never ship a rule the winning
    actor had to break
  or ignore to pass.
STEP 3 -- Port clean-comparison winners. For each CLEAN
   comparison (two trials
  differing on exactly one dimension with a clear winner) whose
    winning value
  differs from the anchor's, edit the anchor to adopt the
    winning value --
  phrased the way the winning trial's skill phrased it, at the
    same level of
  strength. Losing values and dimensions shown to make no
    difference justify
  REMOVING complexity from the anchor, never adding it.
STEP 4 -- Check every sentence you changed or added. For each
   one, name the
  passing trial whose actor actually behaved that way. If none
    exists, the
  edit is your own inference -- revert it. In particular, do not
    introduce or
  strengthen any caution ("leave blank if unsure", "never infer",
    "stop and
    report") beyond what the anchor already contained and its
    passing actor
  actually obeyed: an unverified safety rule is not safe, it is
    untested.
Write the final skill package to:
  environment/skills/final/SKILL.md  [+ optional scripts/
    templates/examples]
Do NOT write variant_1/variant_2 and do NOT write meta.json --
   the final skill is
the deliverable, not an experiment.
A skill is a structured DIRECTORY with progressive disclosure:
   SKILL.md as the
entry point, supporting files for complex operations.
Each SKILL.md must begin with a YAML frontmatter block:
  ---
  name: <skill-name>
  description: <one-sentence description>
  ---
<GENERALIZATION RULE>
<HONESTY RULE>
Task description:
<seed instruction>
```

C.4 SHARED RULES

```
<GENERALIZATION RULE>
IMPORTANT -- Write GENERALIZABLE skills, NOT instance-specific
   recipes:
- Describe general procedures and domain knowledge; do NOT hard-
   code
  specific values, filenames, column names, thresholds, or
    entity names
  taken from the task description.
- A skill that only works for the training instance is WORSE
   than no skill.
- Prefer deterministic, script-based verification (assertions,
   checksums,
  structured diffs) over manually inspecting many images/frames.
    Visually
  reviewing many images inflates the request size and can exceed
    the
  model's context budget, causing the whole rollout to fail.
<HONESTY RULE>
IMPORTANT -- Write skills that reflect TASK REQUIREMENTS, not
   test implementation details:
- Describe what the task actually requires (domain knowledge,
   procedures, constraints).
- Do NOT reverse-engineer test selectors, assertion values, or
   magic strings from execution traces.
- A skill that only passes by matching verifier internals is
   WORSE than no skill.
<VISIBILITY RULE (main experiments)>
Feedback visibility for this ablation:
  The verifier feedback exposed in /root/history contains ONLY
    the trial
  status (PASS or FAIL) and aggregate score. It deliberately
    contains no test
  names, failure causes, traceback, expected values, or verifier
    output.
  Do not infer or claim a failure reason from verifier feedback
    that is not
  present. Use the actor trajectory and the skill package only
    for behavior
  analysis; treat the score as the only verifier signal.
```