UNDERSTANDING AGENT SKILLS: HOW WELL DO THEY TRANSFER TO SMALLER MODELS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Skill evolution gives agents a way to improve over time by turning task-execution experience into reusable procedural guidance. Expressed in text, these lessons can be shared across models without transferring weights. We investigate whether skills developed by frontier models can help much smaller agents generalize to unseen problems while reducing reliance on expensive models. Though text makes this transfer more convenient, smaller models may interpret and follow the same instructions differently from their authors. Across distinct skill-generation frameworks and model families, we find that strong-model skills do not consistently improve smaller executors, and feedback-driven evolution does not consistently outperform direct generation. Analyses of instruction content, model computation, and reasoning trajectories show that responding to a skill does not ensure a better solution. Experiments that introduce guidance at different stages further show that its usefulness can change as a solution develops. Finally, because smaller models are comparatively inexpensive to fine-tune, we compare external skill transfer with internalizing experience through reinforcement learning and on-policy distillation. Training improves performance even without external guidance. After skill-guided training, retaining the document at inference provides further gains, although adding it to a model trained without that guidance can sometimes hurt. Together, these findings show both the promise and the difficulty of transferring experience through skills.

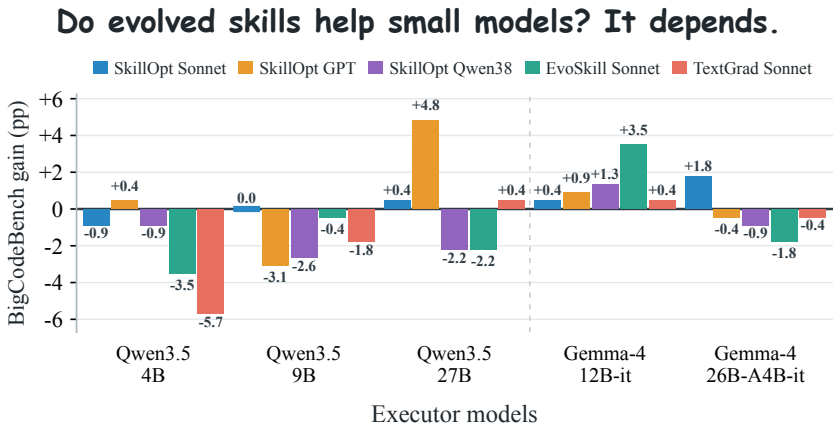


Figure 1: **Key takeaway: a skill's benefit changes with the receiving model.** Small models' gains vary with the skill, task, and executor. The same evolved skill can improve one executor and hurt another. Understanding this phenomenon is critical for making skills developed by strong models useful to smaller ones. Bars show accuracy changes from the no-skill baseline, in percentage points.

1 INTRODUCTION

Agent skills describe procedures that a language-model agent can reuse across tasks. They combine instructions with examples, scripts, or reference material to guide the agent through a class

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.

of problems. A skill can be inspected, revised, and shared with another executor without updating model weights. Adapting model behavior through its inputs has a longer history in automatic prompt optimization (Shin et al., 2020; Zhou et al., 2023). Anthropic introduced Agent Skills in 2025 as discoverable folders of reusable instructions and resources (Zhang et al., 2025). Skill-evolution frameworks subsequently automated their construction and revision using execution experience (Alzubi et al., 2026; Yang et al., 2026; Zhang et al., 2026a). Their reported improvements suggest that external documents can preserve useful lessons and help agents apply them to later tasks.

A practical use of skills is to separate the model that develops a procedure from the model that carries it out. Strong models can explore tasks, diagnose failures, and revise guidance on a sample of problems. Smaller models can then consult that guidance during routine execution, reducing the need to call the stronger model on every task. This division is attractive for cost-conscious deployment, but it asks a skill to carry experience between agents with different capabilities. The receiving model still has to identify relevant advice, interpret its conditions, and carry out the required steps. Moreover, an instruction that is sufficient for its author may leave work unresolved for another executor. Whether a skill transfers therefore depends on what the receiving agent can do with it.

We study this transfer using skills produced by SkillOpt, EvoSkill, and TextGrad (Yang et al., 2026; Alzubi et al., 2026; Yuksekgonul et al., 2025), evaluated on Qwen3.5 and Gemma-4 executors across several model sizes.[1] The tasks cover question answering, mathematics, programming, and interactive household environments. We compare evolved skills with both no-skill execution and skills written directly by the generation model. Direct generation retains access to the strong model's task knowledge without using execution feedback to revise the document. Across these comparisons, the benefit of a skill varies with its source and the executor that receives it. Figure 1 illustrates this on BigCodeBench, where a skill that helps one executor can hurt another, and the relative performance of generation settings changes across models. Feedback-driven evolution also does not consistently improve on direct generation (Table 1). These results raise a question beyond how to construct a better skill: how much of that experience can another agent use?

To understand what happens during transfer, we examine the receiving model along with its final score. We first analyze the procedures encoded in skills and test the effect of removing instruction categories. We then relate these instructions to decisions and semantic similarity in paired reasoning trajectories. Finally, by resuming the same intermediate solution under different instructions, we examine how the stage at which guidance becomes available changes its effect on the outcome. Additional analyses of internal activations and expert selection appear in Appendix B.7.

Training offers another way to transfer useful behavior to a smaller model. Small models are natural targets for fine-tuning because they are cheaper to update and deploy than their larger counterparts. We exploit reinforcement learning and on-policy distillation, comparing training with and without skill guidance and evaluating with and without the skill. Training improves task performance, including in settings where supplying a skill alone provides little benefit. For skill-trained models, retaining the skill at inference provides further gains across our three internalization tasks. These results suggest that training and external guidance can serve complementary roles in adapting smaller agents.

Our study addresses three questions.

RQ1. How well do strong-model skills transfer to smaller executors?
We compare evolved and directly generated skills across tasks and executor sizes to examine whether feedback during skill construction improves their usefulness to the receiving model.

RQ2. How does the receiving model use a skill during problem solving?
We connect instruction content to internal computation and reasoning trajectories, and vary when guidance enters a solution to examine how changes in behavior relate to task outcomes.

[1] We name each skill by its framework and generation model; a single model name means that generation and rollout use the same model. GPT denotes GPT-5.4 (OpenAI, 2026), Sonnet denotes Claude Sonnet 4.6 (Anthropic, 2026b), Opus denotes Claude Opus 4.8 (Anthropic, 2026a), and Qwen38 denotes Qwen3.8-Max (Qwen Team, 2026b). Appendix tables retain explicit generation→rollout pairs where different models are used.

RQ3. What does training add to external skill guidance?
We compare training with and without skills, then retain or remove the skill at inference to assess what the model learns and what external guidance still contributes.

BENEFIT Follow an intermediate relation — **HotpotQA / Qwen3.5-27B**
Skill. "First resolve the bridge entity [. . .], then locate that entity's document to extract the final answer."

Question. At what summer festival might you find music by Bakermat?
Evidence. Bakermat's document says "electronic, deep and tropical house"; the Tropical house document says "Artists of the genre are often featured at various summer festivals such as Tomorrowland."

Baseline ×

"Bakermat is a DJ/electronic music producer. Common summer festivals for electronic music include things like Tomorrowland, Ultra, etc. But I need to find this in the context."

"Based on my searches, **I haven't found a direct connection between Bakermat and any specific summer festival** in the context."

Answer: Information not available in context

Decision. Requires a direct artist–festival link, although the artist's genre is available.

With skill ✓

Before reading. "I need to first read a skill file as required, then search the context for information about Bakermat and summer festivals."

Action: `read_skill_file`
Action Input: `{"path": "skill.md"}`

Later reasoning.
"**I found a connection!** The Bakermat document mentions he makes 'tropical house' music, and the search results show a 'Tropical house' document that mentions artists of the genre are featured at summer festivals such as Tomorrowland. **Let me read the Tropical house document to confirm this connection.**"

Action: `read_context_doc("Tropical house")`
Answer: Tomorrowland

Decision. Uses the genre as the bridge and verifies the second document.

HARM A specific exception becomes a general rule — **BigCodeBench / Qwen3.5-27B**
Skill. "For DB/file-reading tasks that must raise on a missing path, [. . .] raise the spec's exact exception (`ValueError`) BEFORE opening [. . .]."

Task (abridged). Read numeric data from a CSV file, sample 30 values, and return their statistics and histogram.
Exception requirement. The task asks for file-I/O errors to be communicated; its missing-file evaluation expects `IOError`, not `ValueError`.

Baseline ✓

Generated code:
```
except (IOError, OSError) as e:
  raise IOError(f"Error reading file: {e}")
```

Decision. Preserves the file-I/O error type when communicating the failure.

Missing-file evaluation: passes; preserves `IOError`.

With skill ×

Before reading. "Let me start by reading the skill.md file as instructed."

Action: `read_skill_file`
Action Input: `{"path": "skill.md"}`

Generated code:
```
except Exception as e:
  raise ValueError(f"Error reading CSV file: {e}")
```

Decision. Converts every reading error into the exception named in the skill, including a missing file.

Missing-file evaluation: fails; replaces `IOError` with `ValueError`.

Figure 2: **Applying a procedure requires checking where it fits.** Case study shows Qwen3.5-27B executions with initial instructions and SkillOpt Opus. Following a bridge connects two useful facts in HotpotQA. In BigCodeBench, advice to preserve exception types names a specific type that conflicts with this task's evaluation. Quotations and code are verbatim excerpts; decision notes summarize the observed behavior.

2 Related Work

Automatic prompt optimization uses task feedback to revise instructions (Pryzant et al., 2023; Yuksekgonul et al., 2025). Skill evolution organizes execution experience into reusable procedures (Alzubi et al., 2026; Yang et al., 2026). Their usefulness depends on how agents apply them through tools and interaction loops, and whether useful behavior can be retained through training. We discuss prompt optimization, skill construction, agent harnesses, and continual learning in Appendix A.

3 How Well Do Strong-Model Skills Transfer?

A strong model can develop a procedure using its own task executions, then pass the resulting skill to a smaller executor. We evaluate what the receiving model gains from this skill on unseen problems. Comparison with no skill measures its practical benefit; comparison with direct generation asks what execution feedback adds beyond advice the same strong model can already write. Evaluating each skill across executors also shows how its usefulness changes with the model that applies it.

3.1 Experimental Setup

Skills and models. We use SkillOpt, EvoSkill, and TextGrad to construct task-specific skills (Yang et al., 2026; Alzubi et al., 2026; Yuksekgonul et al., 2025). These frameworks share a similar pipeline consisting of two parts: one model acts as a skill curator, which proposes skill revisions using task execution feedback. Another one acts as a skill executor, whose goal is to generate rollouts on corresponding validation sets, giving feedback to the skill curator to improve the skills. The process often cycles in this manner to evolve the skill. The skill curator and skill executor don't have to be the same model, while their weights are usually fixed. The following external-skill comparison uses selected skills generated by newer and stronger models (detailed below). The evaluation executors are Qwen3.5 family (Qwen Team, 2026a) and Gemma-4 family (Gemma Team, 2026) models, including Mixture-of-Experts (MoE) and dense models to ensure generalizability.

Datasets and evaluation. We use four benchmarks spanning reasoning and interactive tasks. HotpotQA (Yang et al., 2018) requires combining evidence across documents to answer multi-hop questions; agents search and read evidence over successive turns. DAPO-Math (Yu et al., 2025) contains competition-style mathematical problems, solved through direct reasoning and answer submission. BigCodeBench (Zhuo et al., 2025) evaluates Python programming with library use; agents write, execute, and revise their solutions. ALFWorld (Shridhar et al., 2021) evaluates household task completion through sequences of actions and environment observations. We use an LLM judge to evaluate answers on HotpotQA and DAPO-Math. BigCodeBench is scored by program execution, and ALFWorld by environment success. More details are provided in Appendix B.3.

Skill generation comparisons. We compare feedback-driven evolution (E) with baseline (no skill) and direct skill generation (D) in Table 1, in which the model writes a skill without task executions or performance feedback. Prompt templates are provided in Appendix C. Each comparison group uses the same generation model in both conditions.

3.2 Results

A useful skill does not help every receiving model. Strong-model skills improve several receiving models. Nevertheless, the gains vary with the skill, task, and executor (Table 1). SkillOpt GPT improves HotpotQA accuracy by 5.9 points on Gemma-4-12B-it but lowers it by 1.2 points on Gemma-4-26B-A4B-it. On BigCodeBench, the same generation setting gives gains of $+0.4$, -3.1, and $+4.8$ points on Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-27B, respectively. We assess agreement between gain patterns using Pearson correlation and Lin's concordance correlation coefficient (CCC), which also accounts for differences in magnitude (Lin, 1989). The cross-executor comparisons show weak agreement on HotpotQA and BigCodeBench (Appendix B.4). More consistent patterns also occur. DAPO-Math has a cross-executor Pearson correlation of 0.985, although its CCC is lower at 0.506 as gains shrink on Qwen3.5-27B. These patterns describe two different outcomes of transfer. On DAPO-Math, skills that help one executor tend to help another, although the size of the benefit changes. On HotpotQA and BigCodeBench, changing the executor can change which document is useful and can reverse the sign of its effect. The BigCodeBench reversal from Qwen3.5-4B to Qwen3.5-9B further shows that transfer quality does not simply increase with executor size. For a strong-model skill to help a smaller agent, that agent must turn the advice into decisions that improve its own solutions. A procedure that is clear to its author may leave intermediate steps implicit or assume judgments that the receiving model does not make correctly. We examine how the receiving models apply the instructions in the following analyses, including cases where an executor follows relevant advice but uses it in the wrong circumstances.

Table 1: Accuracy (%) with evolved (E) and directly generated (D) skills. Bold and underlined values mark the best and second-best scores within each task and executor. Qwen38 denotes Qwen3.8-Max. Avg. is the mean of available executor scores. Results for all four tasks appear in Appendix B.8.

	Qwen3.5			Gemma-4		
Skill	4B	9B	27B	12B-it	26B-A4B-it	Avg.
HotpotQA						
No skill	83.1	81.9	83.9	69.2	**88.4**	81.3
SkillOpt Sonnet (E)	82.1	82.5	<u>84.7</u>	72.0	80.7	80.4
SkillOpt GPT (E)	<u>83.5</u>	**84.3**	**85.0**	<u>75.1</u>	87.2	83.0
SkillOpt Opus (E)	81.9	83.0	83.5	71.4	88.0	81.6
SkillOpt Qwen38 (E)	82.9	<u>84.0</u>	75.0	69.6	82.3	78.8
EvoSkill Sonnet (E)	82.9	83.1	82.9	62.9	86.6	79.7
TextGrad Sonnet (E)	82.4	82.8	73.0	62.8	87.6	77.7
Sonnet (D)	**83.7**	81.9	82.1	61.7	<u>88.2</u>	79.5
GPT (D)	82.8	82.8	82.4	62.9	87.5	79.7
Qwen38 (D)	82.1	81.3	83.6	**84.5**	86.8	83.7
BigCodeBench						
No skill	71.1	75.9	80.3	74.6	<u>79.8</u>	76.3
SkillOpt Sonnet (E)	70.2	75.9	80.7	75.0	**81.6**	76.7
SkillOpt GPT (E)	<u>71.5</u>	72.8	**85.1**	75.4	79.4	76.8
SkillOpt Opus (E)	69.3	**78.1**	<u>82.0</u>	73.2	78.9	76.3
SkillOpt Qwen38 (E)	70.2	73.3	78.1	<u>75.9</u>	78.9	75.3
EvoSkill Sonnet (E)	67.5	75.4	78.1	**78.1**	78.1	75.4
TextGrad Sonnet (E)	65.4	74.1	80.7	75.0	79.4	74.9
Sonnet (D)	<u>71.5</u>	74.1	79.0	74.1	78.9	75.5
GPT (D)	<u>71.5</u>	75.4	81.6	75.4	79.4	76.7
Qwen38 (D)	**72.4**	<u>77.2</u>	79.8	73.2	79.4	76.4

Feedback during construction has uneven benefits after transfer. Directly generated skills offer a useful comparison because they draw on the same strong model's task knowledge without using execution feedback to revise its advice. On BigCodeBench, SkillOpt GPT matches direct GPT generation at 71.5% for Qwen3.5-4B, falls 2.6 points behind it for Qwen3.5-9B, and exceeds it by 3.5 points for Qwen3.5-27B. On HotpotQA, SkillOpt Sonnet trails direct Sonnet generation by 1.6 points for Qwen3.5-4B but improves on it by 0.6 points for Qwen3.5-9B. Revisions informed by the strong model's executions therefore benefit some receiving models without yielding an advantage across executors. The comparison concerns what those agents can use from the revised guidance, beyond what they obtain from directly written advice.

4 How Do Skills Change Problem Solving?

To understand why guidance from a strong model does not reliably improve the receiving model, we follow it from the skill document into the solution process. We first examine what procedures the documents prescribe and whether removing instruction categories changes performance (Section 4.1). Useful advice still has to be applied to the current problem, so we next connect particular instructions to decisions in the trajectories and assess how far the reasoning content departs from ordinary sampling variation (Section 4.2). We then ask whether the advice has a different effect once solving is underway. By holding the preceding work fixed and changing the available instructions at different stages, Section 4.3 tests when the guidance helps or harms. Additional model-internal analyses appear in Appendix B.7.

4.1 What Guidance Do Skills Provide?

We begin by examining which procedures the skills emphasize and how this emphasis changes during evolution.

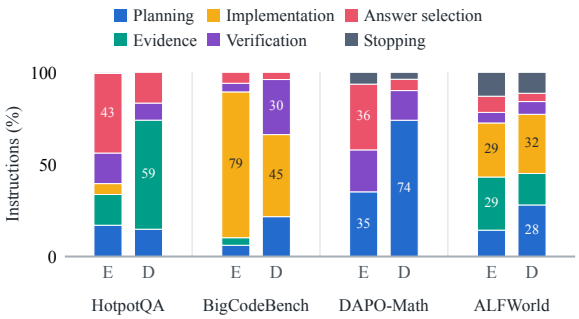


Figure 3: Average instruction shares across all evolved (E) and directly generated (D) skills. Colors denote procedural roles.

What the instructions emphasize. We group instructions by procedural role (Figure 3; criteria in Appendix B.5.1). Their emphasis broadly follows the demands of each task, covering evidence and answer selection in HotpotQA, implementation in BigCodeBench, mathematical methods and checks in DAPO-Math, and state tracking and actions in ALFWorld. Feedback changes the balance among these procedures. From direct generation to evolution, HotpotQA allocates less to evidence gathering (59.2% to 16.7%) and more to answer selection (16.7% to 43.2%); BigCodeBench places more weight on implementation (44.6% to 79.2%). DAPO-Math shifts from planning toward answer selection, whereas ALFWorld increases state-tracking advice (17.1% to 28.9%). The ablations suggest that several parts contribute to a skill's usefulness. Removing a useful category leaves other evolved, task-relevant instructions in place. If those instructions contributed independent benefits, the remaining document could still outperform no skill, even after losing some of the complete skill's advantage. Instead, several deletions reverse that advantage (Table 2). On Qwen3.5-9B, removing implementation instructions lowers BigCodeBench accuracy from 81.58% to 72.81%, below the no-skill score of 75.88%. On Qwen3.5-4B, removing evidence and state-tracking instructions lowers accuracy from 71.05% to 67.54%, also below no skill. Retaining part of an evolved skill therefore need not retain part of its benefit. One possible explanation is that the remaining rules continue to guide decisions after the steps that support or qualify their use have been removed. A shorter skill can then redirect the solution in an unhelpful way rather than simply provide less assistance. Further details are in Appendix B.5.1.

Table 2: BigCodeBench accuracy (%) after deleting instruction categories from SkillOpt Opus. Parentheses give changes from the complete skill, in percentage points.

Instructions	Qwen3.5-4B	Qwen3.5-9B	Qwen3.5-27B
No skill	71.05	75.88	80.26
Complete document	71.05	81.58	85.09
Without implementation	71.05 (0.00)	72.81 (−8.77)	79.82 (−5.26)
Without evidence/state	67.54 (−3.51)	80.70 (−0.88)	82.46 (−2.63)

4.2 How Do Models Use the Guidance?

The content analysis describes what the receiving model is asked to do. We now examine how it uses those instructions in individual decisions and whether the reasoning as a whole changes.

Connecting the rule to the decision. We count the operations addressed by the instructions before examining individual decisions. Across HotpotQA skills and four Qwen3.5 executors, trajectories contain 1.79–2.90 searches and 0.44–1.32 document reads per question; 56.5–68.5% issue a follow-up query after an earlier search or read. Evidence gathering recurs during solving, beyond the initial plan. Cases in Figure 2 show why the applicability of a rule matters. The baseline already considers Tomorrowland but does not verify the relation through the artist's genre. The skill-conditioned trace follows that bridge and checks the second document. In BigCodeBench, the general advice to preserve exception types includes a specific `ValueError` example. That type appears in the skill-conditioned program where the missing-file evaluation requires `IOError`. The skill asks the model to preserve the required exception type, but the model copies its `ValueError` example into a task that requires `IOError`. *The advice itself is sound, but the model can still misapply it to the problem it is solving.*

Table 3: Semantic similarity of cleaned reasoning excerpts. Higher cosine similarity indicates closer semantic content. Baseline resamples compares two independent baseline traces; each skill column averages its similarity to both. Avg. deviation is the mean absolute difference of the two skill columns from Baseline resamples, computed before rounding. Opus is SkillOpt Opus; Sonnet is TextGrad Sonnet.

Task	Qwen3.5	Baseline resamples	Opus	Sonnet	Avg. deviation
HotpotQA	4B	0.884	0.878	0.890	0.006
	9B	0.837	0.856	0.870	0.025
DAPO-Math	4B	0.831	0.842	0.827	0.007
	9B	0.802	0.819	0.814	0.014

Reasoning content remains close to baseline. We next ask whether changes in individual decisions extend to the content of the reasoning as a whole. We remove explicit skill references and recited instructions without rewriting the remaining reasoning, then compare sentence-embedding similarity on the same questions. Independent baseline samples show how much semantic variation arises from sampling alone. Lower skill–baseline similarity would indicate a larger change in reasoning content than this reference. Table 3 shows no consistent decrease; the two skills' average absolute deviation from the reference ranges from 0.006 to 0.025. Across all groups, the latter analysis places skill–baseline similarity within 0.005 of baseline–baseline similarity. The changes in individual decisions therefore coexist with broadly similar reasoning content.

The ablations and trajectory analysis suggest that the value of an instruction depends on the solution it acts on. Advice that is difficult to apply at the outset may become useful once evidence is available. We therefore next vary when the same skill enters the trajectory, comparing earlier guidance with later guidance in an ongoing solution.

4.3 When Guidance Enters the Solution

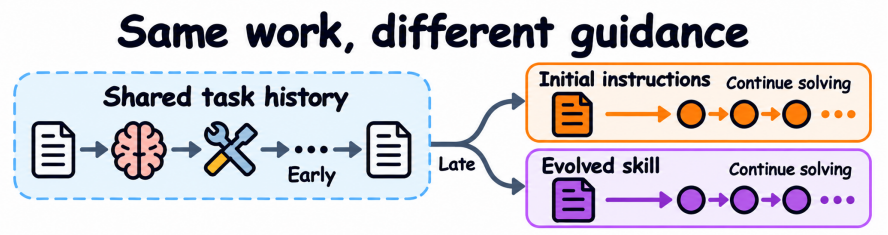


Figure 4: Continuation from a shared task state. At each early or late restart point, both conditions retain the same preceding reasoning, evidence, and tool state. Only the instruction document changes before the model resumes solving.

Do skills mainly change how a solution begins, or can they also redirect work already underway? We resume a baseline trajectory from the same intermediate state under either initial instructions or an evolved skill, keeping its preceding work and remaining budget fixed (Figure 4). Early and late restart points test whether the benefit changes as the solution develops (Table 4; details in Appendix B.6).

Timing changes the value of the same advice. A skill can help revise an existing choice while hindering the process that produces it. On BigCodeBench, all three skills lower Qwen3.5-4B accuracy before code is written but improve it after a program and test feedback are available. Comparing stages on the same questions gives a 9.15–11.97 point advantage to later guidance. HotpotQA shows a similar reversal for GPT and Opus with Qwen3.5-9B, moving from losses of 2.25–2.50 points to small gains. Because evolved skills are distilled from completed attempts, they may be better suited to diagnosing and revising an existing solution than to choosing an initial approach. A candidate program or retrieved evidence provides something concrete to which the advice can apply. Later

Table 4: Continuation accuracy (%) from shared intermediate states. Subscripts give changes from initial instructions in percentage points. Each row holds preceding work fixed across documents. Appendix B.6 defines the Early and Late checkpoints for each task.

Task	Qwen3.5	Stage	No-Skill	GPT	Opus	Sonnet
DAPO-Math	4B	Early	52.00	*Not available*[a]	70.50$_{(+18.50)}$	47.25$_{(-4.75)}$
	4B	Late	53.80	*Not available*[a]	70.11$_{(+16.30)}$	51.90$_{(-1.90)}$
	9B	Early	53.00	*Not available*[a]	68.25$_{(+15.25)}$	56.00$_{(+3.00)}$
	9B	Late	55.43	*Not available*[a]	71.20$_{(+15.76)}$	63.59$_{(+8.15)}$
BigCodeBench	4B	Early	68.24	61.49$_{(-6.76)}$	63.51$_{(-4.73)}$	60.81$_{(-7.43)}$
	4B	Late	72.60	74.66$_{(+2.05)}$	76.71$_{(+4.11)}$	76.71$_{(+4.11)}$
	9B	Early	73.03	70.39$_{(-2.63)}$	71.71$_{(-1.32)}$	70.39$_{(-2.63)}$
	9B	Late	68.92	69.59$_{(+0.68)}$	68.24$_{(-0.68)}$	70.27$_{(+1.35)}$
HotpotQA	4B	Early	93.82	94.89$_{(+1.08)}$	94.89$_{(+1.08)}$	94.09$_{(+0.27)}$
	4B	Late	96.31	95.45$_{(-0.85)}$	96.31$_{(0.00)}$	95.74$_{(-0.57)}$
	9B	Early	88.75	86.50$_{(-2.25)}$	86.25$_{(-2.50)}$	85.75$_{(-3.00)}$
	9B	Late	86.75	87.00$_{(+0.25)}$	87.50$_{(+0.75)}$	86.00$_{(-0.75)}$

[a] GPT already performed well on DAPO-Math during skill generation, and the evolved skill contained few procedural details. We excluded it from these continuation experiments during our manual review of all skills to ensure a reliable comparison of procedural guidance.

guidance is not uniformly better: HotpotQA effects decline with Qwen3.5-4B, and DAPO-Math's Opus gains persist at both stages. Evaluating a skill only at the start misses when its advice becomes useful.

5 From External Skills to Internalized Behavior

Can training retain useful skill-guided behavior without requiring the model to reread the skill? We study Qwen3.5-4B on HotpotQA, BigCodeBench, and ALFWorld using GiGPO reinforcement learning (Feng et al., 2025) and student-top-k on-policy distillation (OPD) (Li et al., 2026c). GiGPO trains on skill-conditioned task executions. OPD uses a frozen Qwen3.5-27B teacher with the skill to supervise continuations generated by the 4B student without it. We also train both methods without the skill, omitting it from GiGPO rollouts or the OPD teacher's context. We evaluate the original and trained models with and without the same document. Appendix B.1.2 gives the objectives and settings.

Table 5: Qwen3.5-4B accuracy (%) before and after training. Rows specify whether training uses a skill; paired columns show inference without or with that skill. Bold and underlined values mark the best and second-best scores in each column. BigCodeBench and ALFWorld report avg@2.

	Inference-time skill					
	HotpotQA		BigCodeBench		ALFWorld	
Training condition	Without	With	Without	With	Without	With
Before training	83.10	83.50	71.05	71.49	62.31	76.49
GiGPO without skill	<u>87.20</u>	89.40	**73.24**	71.93	78.35	80.60
GiGPO with skill	86.90	**90.30**	71.05	<u>74.12</u>	74.25	<u>80.97</u>
OPD without skill	86.40	87.50	72.37	72.81	**82.72**	80.60
OPD with skill	**88.00**	<u>89.90</u>	<u>72.81</u>	**75.88**	<u>78.36</u>	**81.72**

Training changes how the model responds to a skill. Both GiGPO and OPD improve Qwen3.5-4B performance on all three tasks when training and inference both omit the skill (Table 5). Adding a task-relevant skill at inference can nevertheless reduce accuracy. Before training, the skill lowers

scores on HotpotQA and BigCodeBench. The same issue occurs after training without skill guidance: adding the document lowers BigCodeBench accuracy from 73.24% to 71.93% for GiGPO and ALFWorld accuracy from 82.72% to 80.60% for OPD. Relevance to the task therefore does not ensure that guidance fits the model's learned behavior.

After skill-guided training, access to the skill improves both methods on all three tasks, by 3.07–6.72 points for GiGPO and 1.90–3.36 points for OPD. These gains suggest that learning from skill-guided experience may make an agent more receptive to the same advice at inference. The overall pattern is that training improves task performance, while its interaction with external guidance depends on what behavior the model has learned. Useful transfer requires the advice to work with that behavior, beyond simply addressing the right task.

6 Robustness to Stochasticity

To characterize the variability arising from stochastic training and evaluation, we perform repeated runs. Nonetheless, running every configuration multiple times would substantially increase evaluation cost, so we repeated a few selected experiments on HotpotQA and BigCodeBench. For each task, we compare three SkillOpt GPT skills generated from the same start. Cosine similarity of full-document `all-mpnet-base-v2` embeddings shows close semantic agreement (Table 6). We then hold one skill per task fixed and repeat evaluation three times with Qwen3.5-4B and Qwen3.5-27B (Table 7). Accuracy standard deviations range from 0.40 to 1.21 percentage points, and the mean skill gain remains positive for both models on both tasks. These results demonstrate limited variation in generated content and evaluation.

Table 6: Semantic agreement among three SkillOpt GPT skills per task.

Task	Mean similarity
HotpotQA	0.9483
BigCodeBench	0.9084

Table 7: Accuracy (%), mean $\pm$ sample SD over three evaluations of the same skill. BigCodeBench uses avg@2.

Task	Executor	No skill	SkillOpt GPT
HotpotQA	Qwen3.5-4B	83.10 ± 1.21	83.50 ± 0.87
	Qwen3.5-27B	83.87 ± 0.40	85.03 ± 0.75
BigCodeBench	Qwen3.5-4B	71.05 ± 1.16	71.49 ± 0.44
	Qwen3.5-27B	80.26 ± 0.88	85.09 ± 1.16

7 Discussion and Conclusion

Transferring skill's guidance to smaller models offers a practical use of skill's advances. Smaller agents are attractive for low-latency and on-device applications, where repeatedly calling a strong model can be costly or impractical. A strong model could instead develop a skill once and share it with cheaper executors. We study how well this transfer works when the model that writes the instructions and the model that applies them have different capabilities. Across the frameworks and receiving models we evaluate, strong-model skills provide useful gains in some settings but do not consistently improve smaller agents. Feedback-driven evolution also has uneven advantages over direct generation. The receiving models respond to the skills, yet following their advice can produce either useful corrections or inappropriate decisions. The stage at which guidance is introduced changes its effect. Reinforcement learning and on-policy distillation improve smaller-model performance, while external skills remain useful after skill-guided training. Together, these results show that producing a useful procedure and enabling another model to apply it are distinct parts of skill transfer.

Future work will examine whether feedback from smaller executors can help make skills more explicit about intermediate steps that stronger models take for granted. We will also study when to introduce this guidance during execution. Skills distilled from completed attempts may be more useful once an agent has gathered evidence or formed a candidate solution, when the advice can be applied to concrete decisions.

8 AI USE STATEMENT

In this work, we used generative AI tools for assisting in the writing of proofs, supporting qualitative and thematic data analysis, interpreting results, cleaning and reformatting dataset and grammatical refinement toward this manuscript. We have not used generative AI tools for generating synthetic data sets, proposing or refining hypotheses, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation. Additionally, we used generative AI tools for editing software code (debugging), summarizing or analyzing existing literature, discovering research topics or identifying gaps, brainstorming, sourcing/searching for information, editing a research paper to improve readability, identifying relevant literature, formatting references. We have reviewed all AI-assisted work. To be specific, LLM-generated code was verified and tested for correctness and during grammatically refining this manuscript using AI tools, we manually checked contents generated by AI and substantially changed AI-written text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

9 ETHICS STATEMENT

The potential negative social impacts of our method are minor, and align with those typically associated with general LLM reasoning technologies. We emphasise the importance of adhering to the principles of fair and safe deployment of LLM systems, including careful consideration of the training data, transparency about model capabilities and limitations, and ongoing monitoring for unintended consequences.

10 REPRODUCIBILITY STATEMENT

The paper fully discloses all information required to reproduce its main experimental results that underpin the core claims and conclusions of the work. For the proposed method and all baselines, the paper provides complete and clear descriptions of the experimental design, evaluation protocols, and all necessary implementation details to enable independent replication of the results, in full compliance with ICLR reproducibility requirements. For more details, please visit https://anonymous.4open.science/r/Rethinking-Skill-B6B2.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

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

Anthropic. Claude Opus 4.8 system card. System card, Anthropic, 2026a. URL https://www.anthropic.com/claude-opus-4-8-system-card.

Anthropic. Claude Sonnet 4.6 system card. System card, Anthropic, 2026b. URL https://www.anthropic.com/claude-sonnet-4-6-system-card.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Leon Xu, Suzhen Zheng, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. CUA-Skill: Develop skills for computer using agent, 2026. URL https://arxiv.org/abs/2601.21123.

Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. doi: 10.1016/j.patrec.2005.10.010.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, pp. 46375–46408, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Yudong Gao, Zongjie Li, Yuanyuan Yuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. SkillReducer: Optimizing LLM agent skills for token efficiency, 2026. URL https://arxiv.org/abs/2603.29919.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Zhiyuan Jiang, Fangrui Huang, Hanwen Xing, Xander Wu, Yipeng Gao, Rui Cao, Mengdi Wang, Shilong Liu, and Yijiang Li. Demystifying agent skills: Why they work—until they don't, 2026. URL https://arxiv.org/abs/2608.14036.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, 2024. URL https://iclr.cc/virtual/2024/poster/17642.

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, 2026a. URL https://arxiv.org/abs/2602.12670.

Yanchao Li, Wanhao Liu, Ben Gao, Jiaqing Xie, Zhehong Ai, Na Zou, Yuqiang Li, and Tianfan Fu. SkillsInjector: Dynamic skill context construction for LLM agents, 2026b. URL https://arxiv.org/abs/2605.29794.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe, 2026c. URL https://arxiv.org/abs/2604.13016.

Hongxiang Lin, Zhirui Kuai, Erpeng Xue, and Lei Wang. SKILLC: Learning autonomous skill internalization in LLM agents via contrastive credit assignment, 2026. URL https://arxiv.org/abs/2605.27899.

Lawrence I-Kuei Lin. A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, 45(1):255–268, 1989. doi: 10.2307/2532051.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization, 2026. URL https://arxiv.org/abs/2604.02268.

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. GPT-5.4 thinking system card. System card, OpenAI, 2026. URL https://openai.com/index/gpt-5-4-thinking-system-card/.

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

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, pp. 94327–94354, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/980ea04d23d1f6908964eba2a74afe45-Abstract-Conference.html.

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, 2023. URL https://aclanthology.org/2023.emnlp-main.494/.

Qwen Team. Qwen3.5: Towards native multimodal agents. Official model release, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork. Official model release, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, pp. 53728–53741, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/61434.

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

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, 2020. doi: 10.18653/v1/2020.emnlp-main.346. URL https://aclanthology.org/2020.emnlp-main.346/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Eric Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://iclr.cc/virtual/2021/poster/2973.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026a. URL https://icml.cc/virtual/2026/82632.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, 2026b. URL https://aclanthology.org/2026.acl-long.69/.

Zhaoyang Wang, Qianhui Wu, Xuchao Zhang, Chaoyun Zhang, Wenlin Yao, Fazle Elahi Faisal, Baolin Peng, Si Qin, Suman Nath, Qingwei Lin, Chetan Bansal, Dongmei Zhang, Saravan Rajmohan, Jianfeng Gao, and Huaxiu Yao. WebXSkill: Skill learning for autonomous web agents. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026c. URL https://arxiv.org/abs/2604.13318.

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

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, pp. 12028–12068, 2024a. URL https://iclr.cc/virtual/2024/poster/19209.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, pp. 50528–50652, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://iclr.cc/virtual/2023/poster/11003.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, pp. 113222–113244, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://www.nature.com/articles/s41586-025-08661-4.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. Anthropic Engineering, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. In *International Conference on Learning Representations*, pp. 1651–1678, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/06a52a54c8ee03cd86771136bc91eb1f-Abstract-Conference.html.

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

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, pp. 86069–86100, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

Chengfeng Zhao, Yuqiao Tan, Shizhu He, Yequan Wang, Jun Zhao, and Kang Liu. Neural procedural memory: Empowering LLM agents with implicit activation steering, 2026. URL https://arxiv.org/abs/2606.29824.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023. URL https://iclr.cc/virtual/2023/poster/10850.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, pp. 66602–66656, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a6a90bcc2aa470c3871b2d39a67d26e8-Abstract-Conference.html.

APPENDIX CONTENTS

A Additional Related Work **16**

B Detailed Experimental Results **17**

B.1 Methods 17

B.2 Metrics 19

B.3 Additional General Experimental Setup 20

B.4 Gain Consistency 21

B.5 Trajectory Analysis 21

B.6 Shared-Prefix Continuations 23

B.7 Model-Internal Effects of Skills 24

B.8 More Evaluation Results 25

C Prompt Templates **25**

C.1 HotpotQA 25

C.2 BigCodeBench 26

A Additional Related Work

Automatic prompt optimization. Automatic prompt optimization searches for inputs that improve task performance while keeping the underlying model frozen. Early work around 2020 automated discrete prompt search using gradients (Shin et al., 2020); from 2022–2023, researchers increasingly used language models themselves to propose and evaluate natural-language instructions (Zhou et al., 2023; Yang et al., 2024a). Candidate prompts can be generated from demonstrations or revised using a history of prompts and their scores. Feedback on errors offers more specific guidance: ProTeGi uses natural-language critiques to edit instructions, while TextGrad propagates textual feedback through computation graphs to optimize prompts and other components (Pryzant et al., 2023; Yuksekgonul et al., 2025). DSPy extends optimization to programs with multiple model calls by separating program structure from the prompts and demonstrations used by each module (Khattab et al., 2024). Later work like GEPA combines reflection on execution traces with evolutionary search, reporting gains over prompt-optimization baselines and reinforcement learning in its experiments (Agrawal et al., 2026). These works show that optimizing instructions can improve performance on reasoning tasks and multi-stage applications. The updates draw on both the prior knowledge of the proposing model and task feedback.

Skill construction and evolution. Skill libraries allow agents to reuse procedures across tasks (Wang et al., 2024a). Anthropic introduced Agent Skills in 2025 as folders of instructions, scripts, and supporting resources that agents discover and load as needed (Zhang et al., 2025). Recent construction methods use execution experience to decide what these resources should contain. They consolidate recurring lessons into strategy documents or skill directories, sometimes organizing them at several levels, from task plans to individual operations (Zhang et al., 2026b; Ni et al., 2026; Wang et al., 2026a). The iterative methods then evaluate and revise these resources. EvoSkill discovers and refines skill folders through failure analysis and validation-based selection; SkillOpt applies gradient descent to skill documents; CoEvoSkills jointly evolves skill packages and their verifiers (Alzubi et al., 2026; Yang et al., 2026; Zhang et al., 2026a). These methods report gains in knowledge-intensive and interactive tasks, including transfer across models and environments (Ni et al., 2026; Alzubi et al., 2026; Yang et al., 2026). Other construction works connect instructions to executable actions, composing skills through execution graphs etc. (Chen et al., 2026; Wang et al., 2026c). An agent can thus execute a stored procedure or use it to guide its own actions. How skills are presented also affects their utility. Removing non-actionable content and reorganizing supporting material can reduce token cost while preserving or improving performance (Gao et al., 2026); adaptive context construction selects skills, allocates space, and adjusts how they are presented together (Li et al., 2026b). SkillsBench provides diverse tasks paired with curated skills for comparisons across models and harnesses (Li et al., 2026a), and some studies examine the effects of representation, annotation, and retrieval (Jiang et al., 2026). We compare the commonality and transferability of generated skills across frameworks and model scales, and trace how their instructions change reasoning and action on the same problems.

Agents and execution harnesses. An agent's tools and interaction loop determine how it can act on a skill's instructions. ReAct interleaves reasoning, actions, and observations, allowing plans to change as new evidence becomes available (Yao et al., 2023). Work on agent–computer interfaces shows that commands for inspecting files, editing code, and running tests affect software-engineering performance (Yang et al., 2024b). Executable skill systems similarly connect reusable procedures to parameterized actions or combine action programs with step-level guidance (Chen et al., 2026; Wang et al., 2026c). These studies treat the execution harness as part of how an agent uses knowledge. Guidance can also act through internal activations: Neural Procedural Memory derives steering vectors from contrasting trajectories without updating model weights, and reports complementary benefits when combined with textual workflows (Zhao et al., 2026). In agentic search, Search-R1 trains models to generate queries and reason over retrieved evidence through multi-turn reinforcement learning (Jin et al., 2025). This learns how to gather information during problem solving, alongside the ability to reason with the information already in context.

Continual learning. Experience can be retained in an external memory or incorporated into a model's parameters. External approaches store reflections, workflows, and reasoning strategies for reuse across tasks (Shinn et al., 2023; Wang et al., 2024b; Ouyang et al., 2026b). The stored pro-

cedures remain available for inspection and revision, while their use depends on retrieval and the executor's ability to follow them at inference time. Relevant works like SkillOS learn this memory-management process: a curator is trained with reinforcement learning to update an external skill repository, while a frozen executor retrieves and applies its contents (Ouyang et al., 2026a). It reports gains across agentic and reasoning tasks and transfer of the curator across executors. Internalization instead aims to retain useful behavior in the executor itself, so that the skill need not remain in its context. Reinforcement learning methods such as PPO and GRPO, as one approach to internalization, provide reward-based policy updates (Schulman et al., 2017; Shao et al., 2024), while DPO learns from response preferences without separately fitting a reward model (Rafailov et al., 2023). Skill-specific training uses external guidance to support exploration, then progressively withdraws it, or contrasts skill-conditioned and skill-free rollouts to reward autonomous success (Lu et al., 2026; Lin et al., 2026). Other methods jointly train skill generation and use, retaining a library that develops alongside the policy (Wang et al., 2026b; Shi et al., 2026). On-policy distillation (OPD) supplies teacher feedback on the student's own continuations (Agarwal et al., 2024); on-policy self-distillation (OPSD) uses a copy of the model with privileged information as its teacher, supporting new learning while reducing forgetting (Shenfeld et al., 2026). Teacher–student compatibility and the additional knowledge supplied by the teacher affect what is learned (Li et al., 2026c). Our comparison examines whether gains from external skills persist when their guidance is learned through reinforcement learning or distillation and the skill is removed from context.

B Detailed Experimental Results

Sections B.1 and B.2 describe the analysis and training methods and define the evaluation metrics. Section B.3 gives the general experimental setup. Section B.4 examines whether skill gains agree across executors and frameworks. Section B.5 covers instruction categories, category ablations, and trajectory similarity and classification. Section B.6 describes how shared-prefix continuations test the timing of skill guidance. Section B.7 reports the model-internal interventions.

B.1 Methods

B.1.1 Analysis Methods

Activation patching. Activation patching intervenes on intermediate model states while leaving the model parameters fixed (Zhang & Nanda, 2024). Let $h_{\ell,t}^{(0)}$ and $h_{\ell,t}^{(S)}$ denote the activation at layer ℓ and position t under initial instructions and skill S, respectively. At aligned positions, the skill-induced difference and the corresponding additive intervention are

$$\delta h_{\ell,t} = h_{\ell,t}^{(S)} - h_{\ell,t}^{(0)}, \qquad \widetilde{h}_{\ell,t} = h_{\ell,t}^{(0)} + \alpha d_{\ell,t}. \tag{1}$$

We compare the full skill difference, $d_{\ell,t} = \delta h_{\ell,t}$, with an isotropic random direction rescaled to the same norm. Both interventions start from the same reasoning prefix and are followed by a newly generated continuation. Intervention settings are selected on validation questions before evaluation on held-out questions. Appendix B.7 gives the model-specific procedure.

Semantic trajectory representation. We encode retained reasoning text with `all-mpnet-base-v2`, a SentenceTransformers encoder (Reimers & Gurevych, 2019).[2] Each text is split into chunks of at most 384 tokens, including special tokens. We mean-pool non-padding token representations within each chunk and normalize the resulting vector. Chunk vectors are averaged with weights proportional to their content-token counts, then normalized to produce a 768-dimensional text embedding.

N-gram trajectory analysis. We represent a trajectory by counts of contiguous character sequences. For vocabulary $\mathcal{V}$, let $c_g(z)$ be the count of n-gram g in trajectory text z. Its normalized feature vector is

$$\phi(z) = \frac{\big(c_g(z)\big)_{g\in\mathcal{V}}}{\Big\|\big(c_g(z)\big)_{g\in\mathcal{V}}\Big\|_2}. \tag{2}$$

[2] `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`

A logistic classifier predicts whether a trajectory comes from the skill condition, with $y = 1$ for skill and $y = 0$ for baseline:

$$p_i = \sigma\big(w^\top \phi(z_i) + b\big)\,,$$
$$\mathcal{L}_{\text{clf}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log p_i + (1 - y_i)\log(1 - p_i)\right] + \frac{\lambda}{2}\left(\|w\|_2^2 + b^2\right). \tag{3}$$

Here σ is the logistic sigmoid. We pool held-out predictions for evaluation. In cross-skill transfer, the classifier is trained on one skill and evaluated on another.

B.1.2 Internalization Setup and Objectives

Models and skill access. The student starts from Qwen3.5-4B; the OPD teacher is a frozen Qwen3.5-27B. Both use thinking mode. We use the SkillOpt GPT-5.4 document for each task. GiGPO conditions student rollouts on this document. In OPD, the student receives only the short initial instructions. The teacher receives the skill and scores the student's generated tokens on the same task and interaction history.

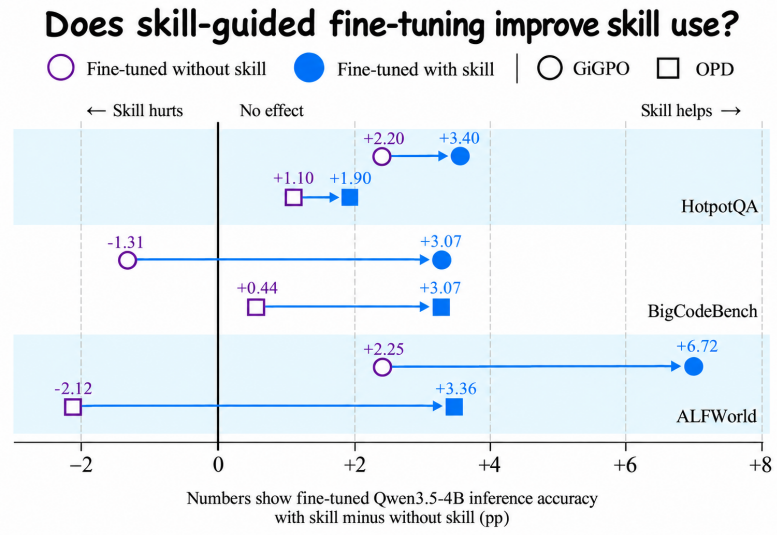


Figure 5: **Inference-time value of a skill after fine-tuning.** Values show Qwen3.5-4B accuracy with minus without the skill at inference, in percentage points. Hollow purple and filled blue markers denote training without and with skill guidance, respectively. Circles indicate GiGPO and squares OPD; arrows connect the two training conditions.

Group-in-group policy optimization (GiGPO). GiGPO combines episode rewards with comparisons among actions taken from matching environment observations (Feng et al., 2025). Within trajectories for the same task and initial state, let R_i be the episode reward and $Q_{i,t}$ the return from action t. We match anchor observations exactly and use mean centering, unit discount, and unit weight for the step advantage:

$$\widehat{A}^{\text{GiGPO}}_{i,t} = (R_i - \overline{R}) + \left(Q_{i,t} - \overline{Q}_{\mathcal{G}(s_{i,t})}\right), \qquad \mathcal{G}(s) = \{(j,u) : s_{j,u} = s\}. \tag{4}$$

Here $\overline{R}$ is the episode-reward mean over collected action records for that task, and $\overline{Q}_{\mathcal{G}(s)}$ is the mean return within an anchor group. The episode mean therefore weights each trajectory by its number of action records. We assign the combined advantage to the generated tokens of the action and optimize a clipped policy objective. Task rewards come from answer scoring, code execution, or environment success, respectively.

The GiGPO runs in Table 5 used BF16 precision during inference and FP32 for training, with learning rates of 10^{-6} on HotpotQA and BigCodeBench and 10^{-7} on ALFWorld.

Student-top-k OPD. We use the probability-weighted, reverse-KL OPD update of Li et al. (2026c). At each student-generated prefix h_t, we retain the student's 16 highest-probability tokens, K_t, and query the teacher on those same candidates. Let $q_{\mathrm{old},t}(v)$ and $p_{\phi,t}(v)$ denote their probabilities given the short instructions and the skill, respectively. The candidate weights and advantages are

$$\begin{aligned} w_t(v) &= \frac{q_{\mathrm{old},t}(v)}{\sum_{u\in K_t} q_{\mathrm{old},t}(u)}, \\ \widehat{A}^{\mathrm{OPD}}_{t,v} &= w_t(v)\left[\log p_{\phi,t}(v) - \log q_{\mathrm{old},t}(v)\right]. \end{aligned} \tag{5}$$

This approximates reverse-KL supervision on student-visited prefixes using a restricted token set. We hold the candidates, weights, and advantages fixed for an update. With $\rho_{t,v} = q_{\theta,t}(v)/q_{\mathrm{old},t}(v)$, the objective is

$$\mathcal{J}_{\mathrm{OPD}} = \mathbb{E}_t\left[\sum_{v\in K_t} \min\left(\rho_{t,v}\widehat{A}^{\mathrm{OPD}}_{t,v}, \mathrm{clip}(\rho_{t,v}, 0.8, 1.2)\widehat{A}^{\mathrm{OPD}}_{t,v}\right)\right]. \tag{6}$$

For negative advantages, dual clipping replaces each summand by its maximum with $3\widehat{A}^{\mathrm{OPD}}_{t,v}$. We sum over candidates and average over assistant tokens. Both models receive the student's preceding reasoning, actions, and tool observations at each turn. Thus teacher feedback also covers the student's intermediate searches and code revisions.

Hyperparameters. Actor parameters and optimizer moments use FP32. AdamW uses $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 0.01, gradient-norm clipping at 1.0, and the constant learning rates in Table 8. Teacher probabilities use temperature 1.0; student sampling uses temperature 0.6, top-$p = 0.95$, top-$k = 20$, and min-$p = 0$. The sampling top-k is distinct from the 16-candidate distillation set. We validate every ten rollout batches and evaluate the checkpoint with the highest validation success rate. Validation uses task scoring without an LLM judge. At test time, each checkpoint uses the same task set and interface with either the short basic instructions (no-skill) or the evolved skill.

Table 8: OPD training and evaluation settings. Token limits are per model call.

	HotpotQA	BigCodeBench	ALFWorld
Learning rate	2×10^{-8}	2×10^{-7}	2×10^{-7}
Prompt / response limit	49,152 / 16,384	49,152 / 16,384	24,576 / 8,192
Context limit	65,536	65,536	32,768
Interaction limit	8	6	50
Test tasks	1,000	114	134

B.2 Metrics

Semantic similarity relative to baseline sampling. For the same question and executor, let B_0 and B_1 be independent baseline traces and S a skill-conditioned trace. With normalized text embedding $e(\cdot)$, we compute

$$\begin{aligned} s_{BB} &= e(B_0)^\top e(B_1), \\ s_{BS} &= \tfrac{1}{2}\left[e(B_0)^\top e(S) + e(B_1)^\top e(S)\right], \\ \Delta_{\mathrm{sem}} &= s_{BB} - s_{BS}. \end{aligned} \tag{7}$$

Higher similarity indicates closer semantic content. Positive Δ_{sem} means that skill–baseline differences exceed those between independent baseline samples; negative values indicate the reverse. We average these quantities over questions and obtain percentile 95% intervals for the mean difference from 10,000 question-level bootstrap samples. These scores compare reasoning content, not solution correctness.

Area under the receiver operating characteristic curve (AUROC). AUROC measures how well a continuous score ranks positive examples above negative examples across decision thresholds (Fawcett, 2006). For score s and threshold τ, the ROC curve plots

$$\mathrm{TPR}(\tau) = \Pr(s \geq \tau \mid y = 1), \qquad \mathrm{FPR}(\tau) = \Pr(s \geq \tau \mid y = 0). \tag{8}$$

Its area can be computed directly from all positive–negative score pairs. With n_+ positive scores s_i^+ and n_- negative scores s_j^-,

$$\widehat{\mathrm{AUROC}} = \frac{1}{n_+ n_-} \sum_{i=1}^{n_+} \sum_{j=1}^{n_-} \left[\mathbf{1}\{s_i^+ > s_j^-\} + \frac{1}{2}\mathbf{1}\{s_i^+ = s_j^-\} \right]. \tag{9}$$

Equivalently, sort scores in ascending order, assign average ranks to ties, and let R_+ be the sum of positive-example ranks:

$$\widehat{\mathrm{AUROC}} = \frac{R_+ - n_+(n_+ + 1)/2}{n_+ n_-}. \tag{10}$$

Sorting gives an $O(N \log N)$ computation. An uninformative ranking has expected AUROC 0.5; perfect positive-over-negative ranking gives 1, and values below 0.5 indicate the reverse ordering. In the trajectory classifier, positive denotes skill-conditioned text, so higher AUROC means greater separability.

Gain agreement across executors and frameworks. For each task, skill s, and executor m, let g_{sm} be accuracy with the skill minus accuracy without it, in percentage points. For two executors, the paired gain vectors X and Y contain the same skill documents. For two frameworks, they contain the same executors while fixing skill construction to Sonnet→Sonnet. Pearson correlation measures whether the gains vary together:

$$r(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X)\,\mathrm{Var}(Y)}}. \tag{11}$$

Lin's concordance correlation coefficient (CCC) also accounts for differences in their means and scales (Lin, 1989):

$$\rho_c(X, Y) = \frac{2\ \mathrm{Cov}(X, Y)}{\mathrm{Var}(X) + \mathrm{Var}(Y) + (\bar{X} - \bar{Y})^2}. \tag{12}$$

All moments use the paired observations and denominator n. CCC equals one when paired gains agree. A high Pearson correlation with lower CCC indicates a similar pattern with a shifted mean or scale. We retain comparisons with at least three shared entries and report the median over eligible pairs. The coefficients describe agreement across the evaluated documents, not independent experimental replications.

B.3 Additional General Experimental Setup

Data splits. Table 9 summarizes the pools used for our experiments.

Table 9: Data pools for skill generation and external-skill evaluation. ALFWorld counts environments; the other tasks count problems.

Dataset	Generation	Validation	Test
HotpotQA	10,000	240	1,000
DAPO-Math	12,393	200	1,500
BigCodeBench	912	114	114
ALFWorld	39	18	134

Execution. HotpotQA agents search and read the context articles supplied with each distractor question. BigCodeBench agents write programs, run visible tests, and revise their code. These evaluations allow six or eight model calls per problem. ALFWorld allows up to 50 environment steps. DAPO-Math uses a single reasoning response. Within each comparison, baseline and skill conditions share the task instructions and execution interface.

Sampling and generation budgets. Qwen3.5-2B, Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-27B use thinking mode, temperature 0.6, top-p 0.95, top-k 20, and min-p 0. Gemma-4-12B-it and Gemma-4-26B-A4B-it use native reasoning and tool calls, temperature 1.0, top-p 0.95, and top-k 64. For both model families, the response limit is 16,384 tokens per call on HotpotQA and BigCodeBench, and 4,096 on ALFWorld. DAPO-Math evaluation uses 16,384 tokens per response. External-skill evaluations use a 65,536-token context window.

B.4 Gain Consistency

Table 10: Agreement in gains over no skill. Entries are median CCC and Pearson r across matched executor or framework pairs. Definitions and comparison sets are in Appendices B.2 and B.4.

	Cross-executor		Cross-framework	
Task	CCC	Pearson r	CCC	Pearson r
HotpotQA	0.005	0.016	0.978	0.979
BigCodeBench	0.017	0.079	0.311	0.777

Table 10 uses the generation comparisons in Table 1, with Qwen3.5-4B, Qwen3.5-9B, Gemma-4-12B-it, and Gemma-4-26B-A4B-it. Metric definitions are in Appendix B.2. Across the four dense Qwen3.5 executors, median cross-executor CCC/Pearson correlations are 0.029/0.312 for HotpotQA, 0.195/0.237 for BigCodeBench, and 0.506/0.985 for DAPO-Math. DAPO-Math's pairwise Pearson correlations range from 0.947 to 0.998, whereas CCC ranges from 0.099 to 0.965.

B.5 Trajectory Analysis

B.5.1 Skill Taxonomy

Instruction categories. With assistance from GPT 5.6 Luna, we label every instruction in the skill depending on its main operation. For example, implementing input validation is an implementation instruction, whereas auditing an assembled program against its contract is verification. Searching again with an alias is evidence gathering; rechecking whether two retrieved facts describe the same person is verification. Annotation is LLM-assisted with source-level review. We first compute each skill's category fractions, then average those fractions within benchmark and E/D group, giving long and short documents equal weight to eliminate length bias. *After automated segmentation, we manually checked the relevant files to ensure their integrity.*

- **Planning**: interpret the task, choose a method or representation, decompose the problem, or order the steps.
- **Evidence**: search, read, link facts, disambiguate entities, or track observed environment state.
- **Implementation**: write code or take an environment action according to its contract, including API choices, return types, and action preconditions.
- **Verification**: test a candidate solution, check factual or logical constraints, diagnose a failure, or revise a failed operation.
- **Answer selection**: identify the requested quantity or entity, extract the answer, or enforce its presentation. Program return types remain implementation rules.
- **Stopping**: decide when to finish, limit repeated work, or allocate a remaining budget.

An excerpt of annotated HotpotQA skill. The intuitive excerpt case below shows how we label the SkillOpt Opus document used in the category ablations. Each original list item is one instruction unit. We label the requested operation rather than the section heading. The excerpts retain their original wording and order.

HotpotQA instruction breakdown **SkillOpt Opus**

Planning

Treat each question as a bridge or comparison problem over the provided context.

Label rationale. This chooses the problem structure before gathering evidence.

Evidence

First resolve the bridge entity (e.g., the singer behind an album, the neighborhood of a person's birth), then locate that entity's document to extract the final answer.

Label rationale. Resolving the entity and locating its document are the main operation. Mentioning the final answer does not make the whole item an answer-format instruction.

Answer selection

Return the shortest answer span supported by the context inside `<answer>...</answer>`.

Label rationale. This specifies which span to submit and how to present it.

Verification

Distractor documents may match one clue or one shared name (e.g., Essex County, Massachusetts when the answer county is English). Verify that the candidate document satisfies ALL the question's named entities/clues; discard any that match only part.

Label rationale. This tests a candidate document against the question's constraints.

Category ablations. In order to gain a deeper understanding of skill taxonomy, we remove blocks assigned to each of the two largest categories in the SkillOpt Opus document, one category at a time. Retained blocks keep their original wording and order. The ReAct budget and sampling parameters are unchanged. Removing evidence/state tracking gives paired 95% intervals of $[-11.40, 4.39]$, $[-7.02, 5.26]$, and $[-7.02, 0.88]$ points for Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-27B.

Table 11: DAPO-Math accuracy (%) for category-deleted SkillOpt Opus skills.

Instructions	Qwen3.5-4B	Qwen3.5-9B	Qwen3.5-27B
No skill	49.07	50.00	85.13
Without planning	81.53	86.60	93.27
Without verification	81.93	87.53	93.40

B.5.2 Classification of Distinct Groups' Trajectories

After cleaning, most skill–baseline comparisons show only modest separation beyond that observed between independent baseline samples (Table 13). On HotpotQA, the two skills yield AUROCs of 0.635 and 0.629 for Qwen3.5-4B, compared with 0.604 for baseline resampling. For Qwen3.5-9B, TextGrad reaches 0.591, below the resampling value of 0.615, while SkillOpt Opus produces a larger difference at 0.696. On DAPO-Math, all four skill comparisons fall between 0.540 and 0.582, against baseline-resampling values of 0.501 and 0.512. Thus, some differences remain in the reasoning after explicit instruction references are removed, but their visibility depends on the skill and executor. The classifier does not find a consistently distinct reasoning style across these comparisons.

Cleaning. Before training a classifier using raw trajectories, we first clean them so that the classifier won't hack by seeing expressions containing "skill", "skillset", etc. An annotator (to be specific, GPT 5.6 Luna) marks explicit skill references, generic instruction recitation, output-format discussion, and standalone repetitions of the question or retrieved evidence for removal. It does not see the condition, generator, or correctness label. Problem-specific plans, calculations, revisions, and checks are retained; a segment that combines substantive reasoning with an instruction reference is retained when deletion would remove the reasoning.

Paired classification. Each skill trace is paired with a baseline trace for the same question. We fit character 3–5-gram counts with a vocabulary of at most 12,000 features, normalize each feature vec-

Table 12: AUROC for distinguishing cleaned reasoning traces. Parentheses give accuracy gains at the same restart stage, in percentage points; green indicates an increase and red a decrease. Opus is SkillOpt Opus; Sonnet is TextGrad Sonnet. Baseline resamples compares independent baseline traces.

Task	Qwen3.5	Baseline resamples	Opus	Sonnet
HotpotQA	4B	0.604	0.635 (+1.08)	0.629 (+0.27)
	9B	0.615	0.696 (−2.50)	0.591 (−3.00)
DAPO-Math	4B	0.501	0.571 (+18.50)	0.540 (−4.75)
	9B	0.512	0.546 (+15.25)	0.582 (+3.00)

tor, and fit an L2-regularized logistic classifier. Five-fold evaluation holds out complete questions, with the vocabulary fitted on each training fold. Baseline-resample comparisons use two initial-instruction samples from the same questions. This comparison estimates how much separability arises from sampling alone, helping account for randomness when interpreting the skill–baseline comparisons. We report pooled held-out AUROC and percentile 95% intervals from 2,000 paired-question bootstrap samples of the fixed held-out predictions (Table 13).

Table 13: Cleaned-reasoning AUROC with paired 95% confidence intervals. Each entry uses n question pairs. Skill columns compare with baseline; Baseline resamples compares two independently generated baseline continuations.

Task	Qwen3.5	n	Baseline resamples	SkillOpt Opus	TextGrad Sonnet
HotpotQA	4B	93	0.604 [0.550,0.658]	0.635 [0.575,0.695]	0.629 [0.575,0.681]
	9B	93	0.615 [0.551,0.676]	0.696 [0.638,0.753]	0.591 [0.528,0.654]
DAPO-Math	4B	100	0.501 [0.447,0.554]	0.571 [0.515,0.624]	0.540 [0.481,0.603]
	9B	100	0.512 [0.456,0.569]	0.546 [0.490,0.604]	0.582 [0.524,0.638]

B.6 Shared-Prefix Continuations

Branch states and sampling. Within a question and model, conditions share the preceding task-specific reasoning, evidence, tool state, and remaining budget. Code tasks also restore the same program. We substitute the instruction document in the context available at the branch, recompute the model state, and generate a new continuation. The preceding reasoning remains fixed rather than being regenerated under the new document. Each executor supplies its own prefixes, so absolute continuation scores across models do not measure from-scratch performance. The early and late states are annotated by task events. For DAPO-Math, they follow an initial plan and an explicit answer candidate. For BigCodeBench, they precede the first program write and follow the first public-test result. For HotpotQA, they follow the first relevant retrieved evidence and retrieved evidence supporting an answer candidate, before final submission. Table 4 reports the two stages separately for all three tasks.

B.7 Model-Internal Effects of Skills

We examine internal responses to the same SkillOpt GPT document on HotpotQA (Figure 6). Following activation-patching methods (Zhang & Nanda, 2024), we inject activation differences into Qwen3.5-4B and Qwen3.5-9B, and remove selected expert contributions in Qwen3.5-35B-A3B. For dense models, we measure the activation difference between skill and initial instructions along the same reasoning prefix, then inject it at continuation. An equal-norm random direction provides a comparison at the same perturbation strength. For MoE, we select experts whose routing weights increase most when the skill is added, and compare their removal with removing experts of similar routing weight. Table 14 also includes equal-norm noise, which retains all expert contributions; None denotes no intervention.

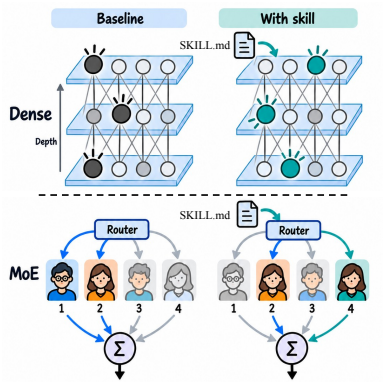


Figure 6: Skill-induced activation and routing changes, illustrated schematically. Table 14 reports the HotpotQA intervention results.

Representation changes and solution quality. Injecting the skill-induced activation difference at the continuation point lowers accuracy by 1.79 points in both dense models. An equal-norm random direction changes accuracy by $+0.89$ points in Qwen3.5-4B and 0.00 in Qwen3.5-9B (Table 14a).

Removing responsive and weight-matched experts. Removing skill-responsive experts lowers evolved-skill accuracy from 75.00% to 73.28% (Table 14b), while initial-instruction accuracy rises from 73.28% to 74.14%. Removing weight-matched experts instead leaves evolved-skill accuracy at 75.00% and raises initial-instruction accuracy to 75.86%. Both interventions reduce the skill's relative advantage by 2.59 points, through different changes in the two instruction conditions. With all expert contributions retained, adding equal-norm noise leaves skill-conditioned accuracy at 75.00%.

Table 14: Accuracy under internal interventions (%). Parentheses in (a) give changes from no intervention. In (b), Δ is the evolved-skill advantage over initial instructions for Qwen3.5-35B-A3B. Differences are in percentage points.

(a) Dense model

Intervention	Qwen3.5-4B	Qwen3.5-9B
None	84.82	79.46
Full skill difference	83.04 (-1.79)	77.68 (-1.79)
Equal-norm noise	85.71 ($+0.89$)	79.46 (0.00)

(b) MoE model

Intervention	Initial	Evolved	Δ
None	73.28	75.00	$+1.72$
Remove responsive	74.14	73.28	-0.86
Remove matched	75.86	75.00	-0.86
Equal-norm noise	76.72	75.00	-1.72

Shared execution states. Execution prefixes are model-specific. Within a model and question, all intervention conditions start from the same prefix and tool state. Each condition has two continuations, scored by answer equivalence and averaged within each question before aggregation. Paired confidence intervals are computed over questions. Intervention settings and expert selection use separate data. Representation injection and expert removal are evaluated against their respective controls.

Representation interventions. For Qwen3.5-4B and Qwen3.5-9B, we measure the activation difference induced by replacing the initial instructions with the evolved skill along a fixed prefix. We inject this difference or an equal-norm random direction at the first continuation, followed by ordinary generation. The full-difference condition injects an activation change rather than presenting the complete skill as a natural input. Both directions are rescaled to the same norm for a given question. Intervention settings are selected on validation data. These interventions follow the use of activation patching to test how internal states affect model outputs (Zhang & Nanda, 2024).

Expert-contribution interventions. For Qwen3.5-35B-A3B, we measure routing-weight changes under the skill on identical continuation tokens from distinct questions. We choose the expert with the largest mean increase in each layer, then retain the four layers with the largest increases. These are the skill-responsive experts. In each selected layer, a different expert is matched by its cumulative routing weights under both initial and evolved instructions, minimizing the sum of absolute differences. These weight-matched experts need not have equal output norms. Expert removal zeros the chosen routing weights without renormalizing the remaining weights. Shared experts remain unchanged. The random-noise condition instead retains all contributions and perturbs the routed output by an isotropic random vector. If h is that output and c the responsive expert's weighted contribution, removal produces $h - c$, whereas the random condition produces $h - \epsilon$ with $\|\epsilon\|_2 = \|c\|_2$. Its norm is matched at each intervened token and layer using the current state. We compare the skill advantage over initial instructions under each intervention to distinguish changes in overall performance from changes in the skill's additional benefit.

B.8 More Evaluation Results

For more of our results, please refer to our anonymized repository `https://anonymous.4open.science/r/Rethinking-Skill-B6B2`, which includes code, some of our evaluation results and notes for reproduction.

C Prompt Templates

The blue boxes show some of the documents used by the baseline. Task and tool-use prompts are shared with the evolved-skill conditions and omitted here. The orange boxes show two matched complete skill cases, which are generated with Sonnet using TextGrad. For more of our results, please refer to our anonymized repository `https://anonymous.4open.science/r/Rethinking-Skill-B6B2`.

C.1 HotpotQA

HotpotQA — **Baseline instructions**

HotpotQA Skill

- Treat each question as a bridge or comparison problem over the provided context.
- Identify the entities and relations needed for both hops before answering.
- Return the shortest answer span supported by the context inside <answer>...</answer>.

HotpotQA — **TextGrad Sonnet**

HotpotQA Skill

Task Type

Multi-hop question answering over 2–10 retrieved documents. Each question requires chaining facts across multiple documents to reach a single, specific answer.

Input Format

Context: [DOC] [TLE] <title> [PAR] <paragraph text> [DOC] [TLE] ...
Question: <question text>

Reasoning Strategy

Step 1 — Identify the bridge entity

Read the question and locate the key relationship it asks about. Most questions require finding an intermediate entity (person, place, event, organization) that connects two facts across different documents.

Step 2 — Trace the hop chain

For each document, extract the specific claim relevant to the question. Build a chain:

HotpotQA **TextGrad Sonnet (continued)**

- Hop 1: Doc A establishes fact X about entity E1
- Hop 2: Doc B connects entity E1 to the final answer

Do not stop at the first plausible fact — verify the chain reaches the exact answer the question requests.

Step 3 — Match the question type

- **Comparison questions** ("which X is larger/older/earlier"): extract the specific attribute from both entities and compare directly. Do not answer with a description of the relationship — answer with the entity itself.
- **Bridge questions** ("who/what is Y of the person who Z"): find the bridge entity first, then look up the attribute of that bridge entity.
- **Set intersection** ("X that is also Y"): find the entity present in both constraint sets.
- **Verification** ("is it true that"): answer yes/no with supporting evidence.

Step 4 — Scope the answer tightly

The answer should be the minimal span that directly answers the question:

- A person's name, not a description of what they did
- A location name, not a geographic description
- A specific number or date, not a range
- A single entity, not a relationship statement like "both finished in the top 15"

Common failure: the question asks *which entity* satisfies a condition, but the answer describes the condition instead of naming the entity. Always re-read the question and confirm the answer type matches what was asked.

Pitfalls to Avoid

- **Wrong hop termination**: stopping at an intermediate entity when the question asks for a property of that entity (e.g., returning a person's name when the question asks for their nickname or award)
- **Plausible but ungrounded answers**: picking an answer that fits the topic but isn't supported by the provided documents — ground every answer in explicit document text
- **Comparison without both values**: for comparison questions, always extract both values from the documents before deciding which wins
- **Over-answering**: returning a full sentence or clause when a short noun phrase is correct

Output Format

Reason silently through the hop chain, then emit only the final answer tag.

Format: <answer>exact answer string</answer>

Examples:

- <answer>Jordi Cruyff</answer>
- <answer>Bomber</answer>
- <answer>Slim Whitman</answer>

C.2 BigCodeBench

BigCodeBench **Baseline instructions**

BigCodeBench Skill

Code Output Discipline

- Read the required starter code and preserve the requested function name and signature.
- Implement the complete function with any imports needed by the starter.
- Return exactly one Python code block containing executable code.

BigCodeBench **TextGrad Sonnet**

BigCodeBench Skill

Objective

Write self-contained Python code that matches the given function signature and passes all tests. Return only the implementation inside a single python ... code block — no explanation, no markdown outside the fences, no extra text.

Core Rules

1. **Use the exact starter code** as your function signature, imports, and constants. Do not rename parameters, add extra imports not in the starter, or change default values unless required.
2. **Return the exact types specified.** If the task says return a tuple, return a tuple. If it says DataFrame, return a DataFrame. Check column names, dtypes, and shape carefully.
3. **Match string literals exactly.** Axis labels, plot titles, column names, error messages, and dictionary keys must match the task description character-for-character, including capitalization (e.g., "Count" not "count", "Item Distribution by Location" not a paraphrase).
4. **Implement all specified exceptions.** If the task says raise ValueError for a condition, raise it. If it says raise TypeError, raise it. Do not silently pass or substitute a different exception type.

Random Seed Handling

- Set the seed **before** any random operations, using the library specified in the imports (e.g., random.seed(seed), np.random.seed(seed)).
- Apply the seed even if it is 0 or None only when not None.

Edge Cases to Handle

- **Empty inputs:** Return an appropriate empty structure (empty DataFrame, empty list, 0, etc.) without crashing.
- **Sample size exceeding data:** When sample_size is larger than available matches or data, return only what is available — do not repeat or pad.
- **Single-element lists/strings:** Handle gracefully; removing the only element should yield an empty string or empty structure as appropriate.
- **Target directory creation:** If the task moves/copies files, create the target directory if it does not exist (os.makedirs(target_dir, exist_ok=True)).
- **NaN handling:** Filter out NaN values before computations unless the task explicitly requires keeping them.
- **Zero-length data:** Return empty results (e.g., shape[0] == 0) when length == 0 or data is empty.

Plotting

- Set axis labels and titles **exactly** as described (copy from task description).
- Use fig, ax = plt.subplots() unless the task requires a specific method.
- For histograms using pandas, use Series.plot.hist(...) as specified.
- Return the Axes object (or Figure + data dict) as the task specifies.
- When plot_path is provided, save the figure with fig.savefig(plot_path).

DataFrame Operations

- Use reindex with fill_value=0 to ensure all specified rows/columns are present even if absent from data.
- When grouping and pivoting, confirm the orientation (rows vs. columns) matches the expected output.
- Column names must match exactly — check for spaces, capitalization, and order.

File and OS Operations

- Always create directories before writing or moving files: os.makedirs(path, exist_ok=True).
- Use re.fullmatch or re.search as appropriate — read the pattern description carefully. re.search finds a match anywhere; re.fullmatch matches the whole string.
- When iterating directory contents, check os.path.isfile before operating on entries.

String Manipulation

- When removing a comma-separated value from a string, split on ', ' (with space) unless the task specifies otherwise, and rejoin the same way.
- When the task says "remove a substring", the original string is preserved in the output; only the modified version changes.
- Always append the original (unmodified) string to the output, not a cleaned/stripped version.

BigCodeBench **TextGrad Sonnet (continued)**

Statistical Functions

- For t-tests and similar: validate inputs (non-empty, $\geq$2 non-NaN values, sufficient variance) and raise ValueError with a descriptive message before computing.
- Use nan_policy='omit' where specified.

Verification Checklist Before Finalizing

- [ ] Exact function signature from starter code
- [ ] All return types and shapes match the description
- [ ] All string literals (labels, titles, column names) copied verbatim from the task
- [ ] All required exceptions are raised for the specified conditions
- [ ] Edge cases handled (empty input, oversized sample, missing directories, single-item lists)
- [ ] Random seed set correctly using the right library
- [ ] No extra output, print statements, or markdown outside the code block