SKILLREIN: LEARNING REUSABLE AGENT SKILLS WITH REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement calls for agents that improve both task execution and the mechanisms that support future improvement. Reusable natural-language skills offer a practical interface, yet plausible guidance can be difficult for the receiving agent to apply. We introduce SkillRein, a reinforcement learning framework that learns to generate shared skills from execution experience. SkillRein trains a dedicated proposer while keeping the task executor frozen. The proposer reads selected successful and failed attempts and produces a set of procedures for related tasks. An outcome-and-use objective evaluates the proposed set through paired task outcomes and reported skill applications, training the generation mechanism against downstream utility. This design learns how to produce guidance while expressing behavioral adaptation through external language. Across four categories of search tasks, a small proposer trained with SkillRein produces more useful guidance for the executor it is trained with than its initialization and much larger models, and its guidance completes tasks with fewer actions. The same proposer also improves a larger executor that it never saw during training, and on WebShop, learning likewise improves the proposer's guidance over its initialization. The learned skills let the executor finish tasks within its action limit. These results establish learned skill generation as a useful component of agent self-improvement and motivate learning guidance together with its downstream use.

1 INTRODUCTION

Recursive self-improvement (RSI) asks agents to improve not only their behavior but also the mechanisms through which they acquire better behavior. Recent work on self-modifying agents and experience-driven learning makes parts of this ambition experimentally accessible (Zhang et al., 2026a; Xia et al., 2026). Reusable natural-language skills offer a practical interface. A skill states a procedure and the conditions under which it should be applied, allowing an agent to adapt through external guidance while retaining its parameters. The central challenge is to learn how to turn execution experience into procedures that the receiving agent can use effectively. Improving this experience-to-skill mapping makes the mechanism of adaptation itself a target of learning.

Existing approaches organize experience into reflections, executable libraries, or contextual playbooks (Shinn et al., 2023; Wang et al., 2024; Zhang et al., 2026b). Prompted generators are not trained against the downstream utility of their advice, and plausible guidance can be too specific or difficult for the receiving agent to apply. Learning feedback for a frozen actor addresses this mismatch (Yao et al., 2024), but does not by itself define how to score a shared set of procedures across tasks. Recent RL methods address skill generation through recurrent instance-level revision, anchor-query editing, or joint evolution of skills and task policies (Vishe et al., 2026; Shen et al., 2026; Zhang et al., 2026c). Instance-level feedback does not directly measure the usefulness of a complete skill set across a task batch, while joint policy training changes the executor that the skills must guide. The problem we address is learning to generate one shared set of procedures from batch-level experience for a fixed executor. This requires feedback that connects proposed procedures to both their application across related tasks and the resulting changes in task outcomes.

We introduce SKILLREIN, a reinforcement learning framework that trains a dedicated skill proposer through a frozen executor. Figure 1 shows the learning process. The executor first attempts

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

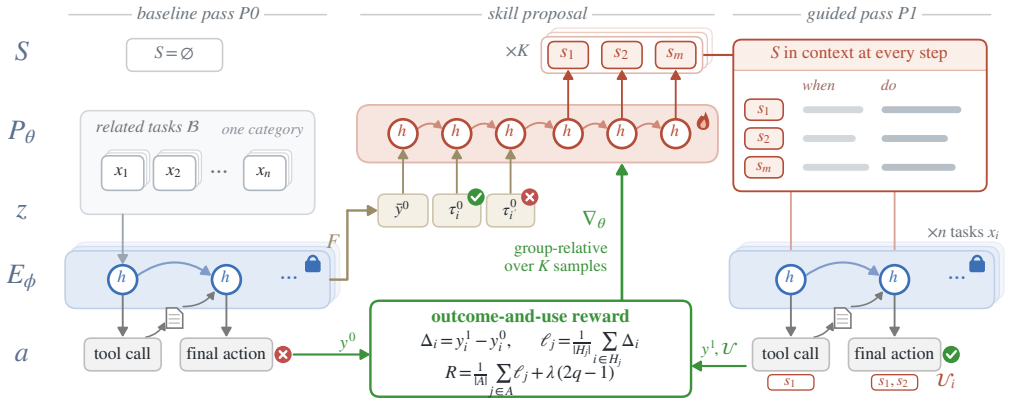

Figure 1: **Architecture of SKILLREIN.** A frozen executor E_ϕ first attempts a batch $\mathcal{B}$ of n related tasks without skills (left). The formatter F turns the baseline success rate $\bar{y}^0$ and excerpts of successful and failed trajectories into experience z, from which the trainable proposer P_θ decodes K candidate skill sets (center). Each set S, whose entries pair an identifier with a condition and a procedure, stays in the executor's context at every step, and the executor reports the skills $\mathcal{U}_i$ it applies (right). The outcome-and-use reward averages the paired outcome changes Δ_i over the tasks citing each skill and updates only the proposer through group-relative advantages over the K sets. At test time, a new batch passes through the same computation once without the update.

a batch without skills, and the proposer reads selected successful and failed executions to generate a common set of procedures. The same executor then revisits the batch with these skills. We score each reported skill through paired outcome differences on the tasks that cite it, aggregate these scores across used skills, and reward coverage of the proposed set. This objective trains the proposer to produce guidance whose value is measured through downstream execution. Across training episodes, the proposer learns how to distill experience; within an episode, the executor adapts through the resulting language. Because the proposer learns only from the executor's trajectories and pass rates in the environment, the same loop can keep refining skill generation as the executor grows stronger or the environment changes, turning the mechanism of adaptation itself into a target of recursive self-improvement.

We evaluate SKILLREIN on search questions from seven question-answering datasets with Qwen3-4B and Qwen3-8B executors and on web shopping in WebShop. In the main evaluation (Table 1), the trained 4B proposer raises the success of the Qwen3-4B executor from 30.9% to 36.5% and improves every task category, whereas five much larger models used as proposers, including two proprietary ones, reduce it to between 11.4% and 20.7%. Reinforcement learning lifts guided success far above the 20.9% of the untrained proposer, and the learned skills also raise the success of the Qwen3-8B executor, which was not used in training, from 18.6% to 30.1% (Figure 2a). On WebShop, the learned skills raise the executor's test success from 3.3% to 17.7%. The learned skills also let the executor finish within its action limit, so guided execution needs fewer searches and tokens than with the skills of the large models.

Our contributions are threefold.

- We formulate the generation of a shared skill set as a learnable component of agent self-improvement, with one proposal serving multiple tasks executed by a fixed agent.
- We develop an outcome-and-use objective that scores proposed skills through reported applications and paired task outcomes, directly training the experience-to-skill mapping.
- We show on search and WebShop tasks that learning makes a small proposer's guidance more useful than that of its initialization, and on search that the learned proposer produces more useful and cheaper-to-execute guidance than much larger models for its receiving executor, with guidance that fits the executor's action limit learned from downstream reward alone.

2 METHODOLOGY

SKILLREIN learns to turn experience into guidance that a fixed executor can apply across related tasks. Its action is a complete shared skill set, and its feedback combines outcome changes with reported applications of the proposed procedures. Figure 1 separates skill generation, task execution, and proposer learning.

2.1 LEARNING SHARED SKILLS FOR A FROZEN EXECUTOR

Let E_ϕ be an executor with fixed parameters and $\mathcal{B} = \{x_i\}_{i=1}^n$ a batch of tasks from a common category. The executor interacts with tools to produce a trajectory τ_i and receives a binary correctness label y_i. A skill is a natural-language procedure with an explicit applicability condition. The proposer P_θ produces a set $S = \{s_j\}_{j=1}^m$, which is inserted into the context of every task in the batch. Thus, one proposal influences n executions, each with its own observations and action sequence.

Training begins with a baseline pass, denoted P0, in which the executor attempts the tasks without skills. This pass defines the experience that the proposer must learn from,

$$D^0 = \{(x_i, \tau_i^0, y_i^0)\}_{i=1}^n, \qquad \tau_i^0 \sim E_\phi(\cdot \mid x_i, \varnothing), \qquad z = F(D^0), \tag{1}$$

where F selects and formats execution records. After observing z, the proposer samples $S \sim P_\theta(\cdot \mid z)$. The guided pass, P1, uses the same executor, tasks, tools, and action budget,

$$D^1 = \{(x_i, \tau_i^1, y_i^1, \mathcal{U}_i)\}_{i=1}^n, \qquad \tau_i^1 \sim E_\phi(\cdot \mid x_i, S). \tag{2}$$

Here $\mathcal{U}_i$ contains the skill identifiers reported during task i. The proposer learns from the utility of its entire set under this receiving executor,

$$\max_\theta J(\theta) = \mathbb{E}_{\mathcal{B}, D^0, S \sim P_\theta(\cdot \mid F(D^0)), D^1} \left[R(D^0, D^1, S) \right], \qquad \phi \text{ fixed.} \tag{3}$$

This formulation provides two forms of reuse. The learned proposer maps different execution batches to guidance, and each generated set serves multiple related tasks. When the executor or execution budget changes, its own baseline experience conditions a new proposal. The skill-generation policy carries learned experience across episodes, while the skills adapt its guidance to the current agent and task batch.

Algorithm 1 SKILLREIN proposer training

Require: Frozen executor E_ϕ; proposer P_θ; task-batch distribution; K proposals per group

for each training iteration **do**
  Sample groups of related-task batches $\{\mathcal{B}_g\}$
  for each group g and candidate $k = 1, \ldots, K$ **do**
    Obtain baseline experience $D_{g,k}^0$ from E_ϕ on $\mathcal{B}_g$
    $z_{g,k} \leftarrow F(D_{g,k}^0)$ ▷ Selected successes and failures
    Generate $S_{g,k} \sim P_\theta(\cdot \mid z_{g,k})$ ▷ One shared set
    Apply $S_{g,k}$ to $\mathcal{B}_g$ with E_ϕ; collect $D_{g,k}^1$
    Compute paired outcome differences and reported skill applications
    $r_{g,k} \leftarrow R(D_{g,k}^0, D_{g,k}^1, S_{g,k})$ using Eq. (6)
  end for
  Update θ by RL using proposal rewards $\{r_{g,k}\}$; keep ϕ fixed
end for
return learned proposer P_θ

2.2 EXPERIENCE-CONDITIONED SKILL GENERATION

Representing successes and failures. The formatter F provides the task category, baseline success rate, action budget, and excerpts of actions and observations from selected trajectories. It takes up to the first four successful and six failed records in their original order. An unavailable quota remains unfilled. Each retained record includes the question and correctness label. For each recorded step, the formatter retains the first 300 characters of the normalized action field and 180 characters of the observation field. These bounded excerpts condition the proposer on the available execution record, together with explicit feedback on success and failure. The representation is shared by all compared proposers; Appendix B details its construction.

A policy over procedural guidance. Each skill contains an identifier, a title, a principle, and a condition specifying when it should be applied. The prompt requests category-level guidance and approximately five entries. If $b_{1:V}$ is the serialized response, the proposal policy factorizes as

$$P_\theta(b_{1:V} \mid z) = \prod_{v=1}^{V} P_\theta(b_v \mid z, b_{<v}), \qquad S = \mathrm{parse}(b_{1:V}). \tag{4}$$

We initialize this policy directly from Qwen3-4B (Yang et al., 2025) and update its full parameters by RL, without an additional supervised fine-tuning stage. The executor receives the same generated set for all tasks in $\mathcal{B}$. A common wrapper presents nonempty skills as mandatory operating procedures, asks the executor to consult them before each action, and requests applied identifiers in a `<used_skills>` field. This wrapper connects the textual interface to the training signal and is held fixed across skill proposers. For example, a recorded skill, *Step Necessity Check*, instructs the executor to stop searching when existing evidence already satisfies the question. Its applicability condition asks the agent to reassess the remaining information gap after each search. Such a procedure can govern many trajectories without specifying any task's answer; Appendix B gives the complete entry.

Learning across proposal episodes. Algorithm 1 evaluates candidate sets on their associated baseline batches. The proposer makes one outer action per episode, while the executor's tool interactions remain inside the environment that scores that action. Guided trajectories provide the reward for updating P_θ; the next episode starts from baseline experience on its sampled batch. Learning therefore accumulates in the skill-generation policy, with behavioral adaptation expressed through the skills it produces. At test time the same episode runs once without the update. The executor collects baseline experience on a new batch, the learned proposer writes one shared set, and the executor solves the batch with it. Appendix A specifies the optimizer and recorded training configuration.

2.3 Outcome-and-use feedback

Scoring the procedures the executor applies. For each paired task, let $\Delta_i = y_i^1 - y_i^0$ be its change in correctness. Define the reported-use matrix $U_{ij} = \mathbb{I}[\mathrm{id}_j \in \mathcal{U}_i]$, the task set $H_j = \{i : U_{ij} = 1\}$ for skill j, and the used-skill set $A = \{j : |H_j| > 0\}$. The skill-conditioned lift and its aggregate are

$$\ell_j = \frac{1}{|H_j|} \sum_{i \in H_j} \Delta_i \quad (j \in A), \qquad L = \frac{1}{|A|} \sum_{j \in A} \ell_j, \qquad q = \frac{|A|}{m}. \tag{5}$$

The lift measures outcome changes on tasks reporting a skill, and L gives equal weight to each used skill. Coverage q measures how much of the proposed set is reported as applicable at least once. The proposal reward is

$$R(D^0, D^1, S) = c\,[L + \lambda(2q - 1)], \qquad c = 1, \quad \lambda = 0.05. \tag{6}$$

Outcome lift supplies the main utility signal, while the coverage term encourages the proposer to populate its set with applicable procedures. With $L \in [-1, 1]$, coverage adjusts the reward by at most 0.05 in either direction. We use $L = 0$ when no aligned application exists; Appendix C specifies empty proposals and alignment.

How shared skills determine the training signal. For a nonempty used-skill set, the lift can equivalently be written as a weighted sum of paired task outcomes,

$$L = \sum_{i=1}^{n} w_i \Delta_i, \qquad w_i = \frac{1}{|A|} \sum_{j \in A} \frac{U_{ij}}{|H_j|}, \qquad \sum_{i=1}^{n} w_i = 1. \tag{7}$$

Each used skill receives equal weight, distributed over the tasks that cite it. A broadly cited procedure therefore does not dominate solely because it appears in more trajectories; a selectively applied procedure can contribute equally to the set's score. Multiple procedures can place weight on the same task. This gives the proposer feedback at the level of its reusable guidance while still measuring downstream task outcomes. Reported use determines the aggregation weights, rather than identifying an isolated causal effect of each skill. Evaluation measures success across all attempted tasks. The reward in Eq. (6) trains the complete proposal as a single language-model action, keeping the executor fixed throughout.

3 EXPERIMENTS

We evaluate SKILLREIN in three stages. First, we detail the experimental setup, including datasets, baselines, evaluation protocol, and metrics (Section 3.1). Next, we compare the learned proposer with its initialization, direct experience memory, and five large-model proposers (Section 3.2). Finally, we study transfer across executors, execution efficiency, WebShop, learning dynamics, much larger executors, and robustness across generations (Section 3.3). Supplementary results appear in Appendices E and F.

3.1 EXPERIMENTAL SETUP

Datasets and splits. We evaluate SKILLREIN on search-based question answering and web shopping. The search benchmark comprises seven datasets, namely NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2023), HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). Questions are grouped into four task categories (direct retrieval, multi-hop reasoning, entity-attribute lookup, and comparison), and each batch contains 50 questions from one category. The search proposer is trained on NQ and HotpotQA training questions. Our main search evaluation (Table 1) uses 64 batches sampled from this training question pool, 16 per category, with 3,200 question occurrences and 2,617 distinct question texts. For WebShop (Yao et al., 2022), a separate proposer is trained on synthetic instructions over a development catalog of 1,000 products and evaluated on six test batches of 50 instructions, one per product category. Appendix D details the search data scope, and Appendix F describes the WebShop split.

Models and baselines. We initialize the skill proposer from Qwen3-4B (Yang et al., 2025) and train its full parameters with reinforcement learning, without an additional supervised fine-tuning stage. The search proposer is trained through a frozen Qwen3-4B executor, and Qwen3-8B serves as an additional receiving executor without further proposer training. WebShop uses a separately trained Qwen3-4B proposer and a frozen Qwen3-4B executor. We compare against three types of reference conditions: (i) *no skills*, where the executor solves tasks without additional guidance; (ii) *experience memory*, where the selected baseline execution records are placed directly in the executor's context; and (iii) *skill proposers*, namely the untrained Qwen3-4B initialization and five large-model proposers, the proprietary Opus 5 and GPT-6 Astra and the open GLM-5.2, DeepSeek V4 Pro, and Qwen3.5-397B-A17B. On WebShop, we compare the learned proposer with execution without skills. Appendices A and F give the training configurations.

Evaluation protocol. For each batch, the executor first attempts all 50 questions without skills, and every proposer receives the same formatted experience, including the category, baseline success rate, action budget, and up to four successful and six failed trajectories. Each proposer writes one shared skill set, which the executor then applies to all 50 questions through the same mandatory skill-use wrapper; unparsable proposals fall back to execution without skills and remain in the results. Search runs in the Search-R1 environment (Jin et al., 2025) with the Wiki18 corpus (Karpukhin et al., 2020), the E5-base-v2 retriever (Wang et al., 2022), three passages per search, and greedy executors with a budget of four actions; WebShop uses the same protocol with a ten-action budget. Search results use the released `search_200` checkpoint and the first generation of each method, except the Qwen3-8B success in Figure 2a, which comes from an earlier run on the same batches with the proposer served from the training run (Appendix D).

Metrics. We evaluate downstream task performance and execution cost.

- **Search success.** Exact-match accuracy over all attempted question occurrences, in percent; questions left unanswered at the action limit count as failures. Category columns are computed within each category, and the Overall column pools every occurrence rather than averaging category accuracies.
- **WebShop success.** The percentage of instructions that end in a purchase with terminal reward 1.0, pooled over the evaluated test batches.

Table 1: **Main results** with the Qwen3-4B executor: search success (%) on 64 batches of 50 questions by question category. Parentheses give changes from execution without skills, and bold marks the best proposer in each column. All methods receive the same baseline experience.

Method	Direct retr.	Multi-hop	Entity attr.	Comparison	Overall
No skills	33.3	32.5	27.6	30.1	30.9
Experience memory	32.6 (−0.7)	32.4 (−0.1)	28.8 (+1.2)	30.4 (+0.3)	31.0 (+0.1)
Base Qwen3-4B	26.3 (−7.0)	22.4 (−10.1)	19.8 (−7.8)	15.1 (−15.0)	20.9 (−10.0)
Opus 5	9.3 (−24.0)	14.3 (−18.2)	12.1 (−15.5)	10.1 (−20.0)	11.4 (−19.5)
GPT-6 Astra	19.6 (−13.7)	24.0 (−8.5)	19.5 (−8.1)	19.6 (−10.5)	20.7 (−10.2)
GLM-5.2	16.6 (−16.7)	23.3 (−9.2)	17.8 (−9.8)	14.0 (−16.1)	17.9 (−13.0)
DeepSeek V4 Pro	18.8 (−14.5)	19.5 (−13.0)	13.5 (−14.1)	15.4 (−14.7)	16.8 (−14.1)
Qwen3.5-397B-A17B	14.5 (−18.8)	23.0 (−9.5)	16.6 (−11.0)	17.3 (−12.8)	17.8 (−13.1)
SKILLREIN	**39.9** (+6.6)	**40.8** (+8.3)	**32.6** (+5.0)	**32.6** (+2.5)	**36.5** (+5.6)

- **Execution cost.** The mean number of executor searches and of executor tokens, input plus output, per attempted task, including failures. Costs refer to guided execution; Appendix C accounts for baseline collection and skill generation.

3.2 MAIN RESULTS

Main comparison. Table 1 compares the learned proposer with execution without skills, experience memory, its untrained initialization, and five large-model proposers, using the Qwen3-4B executor it was trained with. The learned proposer achieves the highest success in every category, raising overall success from 30.9% to 36.5%, whereas every other skill proposer lowers it. The key findings are as follows.

- **Finding 1. A small trained proposer produces the most useful guidance.** The learned skills improve every category, by 8.3 points on multi-hop reasoning, 6.6 on direct retrieval, 5.0 on entity-attribute lookup, and 2.5 on comparison questions. In summary, the trained 4B proposer writes guidance its executor can use for every question type.
- **Finding 2. Larger models write guidance that does not fit the executor.** The skills written by the five large models lower success in every category, by 8.1 to 24.0 points. Opus 5 gives the lowest overall success, 11.4%, and GPT-6 Astra the highest, 20.7%, with the three open models in between at 16.8% to 17.9%. In summary, neither scale nor proprietary training yields procedures that fit the executor.
- **Finding 3. Learning, rather than experience alone, drives the gain.** Placing the selected experience directly in the executor's context leaves success at 31.0%, close to execution without skills, and the untrained Qwen3-4B proposer lowers it to 20.9%. RL on the same model raises success to 36.5%, 15.6 points above its initialization, with gains in every category. In summary, since the executor is frozen throughout, the improvement comes from what the proposer learns to write.

3.3 FURTHER ANALYSIS

Transfer across executors. The proposer is trained through the Qwen3-4B executor, so we ask whether the skills it writes also serve an executor it never saw during training. We apply the same proposer, without further training, to Qwen3-8B, which collects its own baseline experience on the same batches, and compare it with GLM-5.2, DeepSeek V4 Pro, and Qwen3.5-397B-A17B as proposers. The learned skills raise the success of Qwen3-8B from 18.6% to 30.1%, above the 26.3% to 27.4% reached with the three large models (Figure 2a), and give the highest success in three of the four categories, with the largest gain, 13.4 points, on multi-hop reasoning (Appendix E). The skills of the large models help the 8B executor but harm the 4B executor, whereas the learned skills help both.

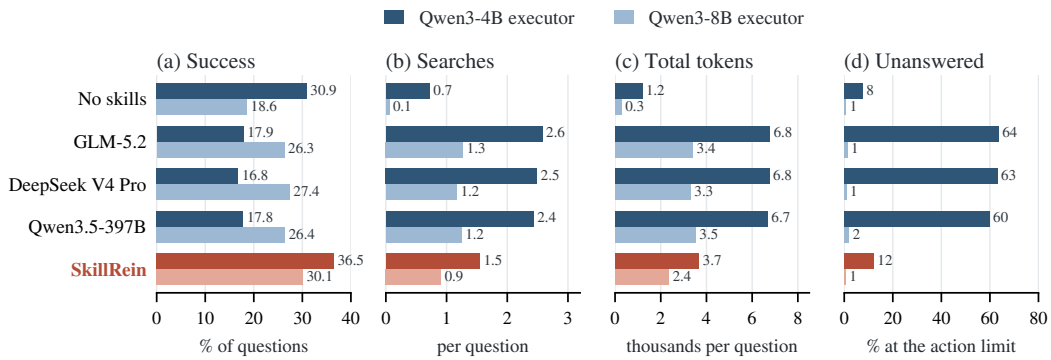

Figure 2: **Transfer and efficiency.** (a) Success with the Qwen3-4B executor, as in Table 1, and with the Qwen3-8B executor, as in Table 6. (b) Searches, (c) executor tokens, input plus output, and (d) the share of questions left unanswered at the four-action limit, per question.

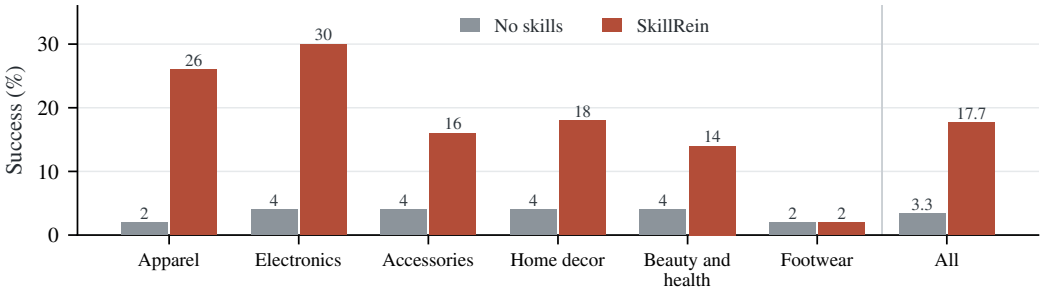

Figure 3: **WebShop.** Test success (%) of the frozen Qwen3-4B executor without skills and with skills from the learned WebShop proposer, on 50 instructions per product category.

Execution efficiency. Figure 2b, c, and d show how skills change the executor's effort, as searches, executor tokens, and the share of questions left unanswered at the four-action limit, per question. With the Qwen3-4B executor, the learned skills use 1.5 searches and 3.7k tokens per question and leave 12% of questions unanswered, whereas the skills written by the three large models prescribe 2.4 to 2.6 searches, cost 6.7k to 6.8k tokens, and leave 60% to 64% unanswered. With the Qwen3-8B executor, the learned skills also use the fewest searches and tokens among the skill proposers, 0.9 searches and 2.4k tokens against 1.2 to 1.3 searches and 3.3k to 3.5k tokens. The learned procedures thus fit the executor's action budget, making guidance both more effective and cheaper.

WebShop. To test the approach beyond question answering, we train a separate proposer in WebShop, where the executor must search a product catalog and buy an item that matches an instruction within ten actions, and evaluate it on test instructions from six product categories (Figure 3). Without skills, the frozen Qwen3-4B executor rarely completes a matching purchase and succeeds on 3.3% of the instructions. Skills from the learned WebShop proposer raise success to 17.7%, with gains in five of the six categories, led by electronics, from 4% to 30%, and apparel, from 2% to 26%. During training, validation success with the proposer's skills rises from 1.0% at initialization to 18.3% (Appendix F), so the gain comes from learning rather than from the skill interface itself.

Learning dynamics. To see how the proposer acquires this behavior, Figure 4 traces the training run of the search proposer. The outcome-and-use reward on training batches starts negative, because the untrained proposal policy harms the executor, turns positive within the first hundred steps, and continues to rise. On the same batches, guided success rises in parallel from 19% at initialization, overtakes execution without skills near step 70, reaches 36% at the released checkpoint, and keeps rising to about 38% by step 400, while the executor alone stays near 30%. Since the executor is frozen throughout, these gains come entirely from what the proposer learns to write. The reward

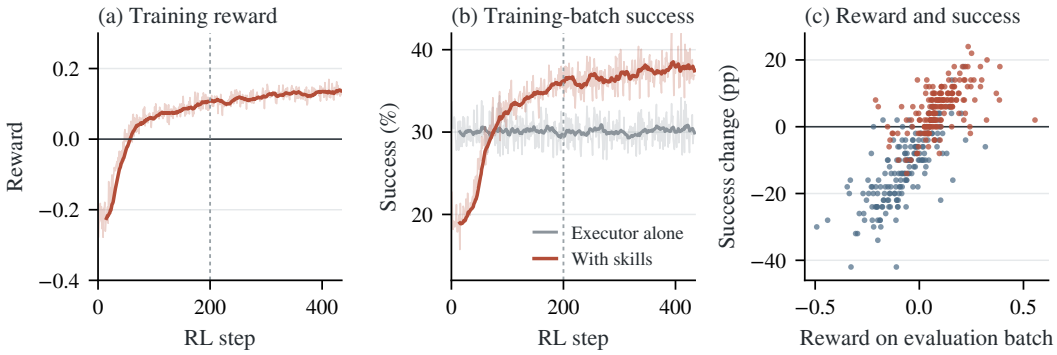

Figure 4: **Learning dynamics.** (a) Outcome-and-use reward on training batches and (b) success of the frozen Qwen3-4B executor on the same batches, alone and with skills from the current proposer (15-step moving averages). The dashed lines mark the released step-200 checkpoint. (c) The same reward for base (blue) and learned (red) skill sets on the batches of Table 1 against their success change with the Qwen3-4B executor; empty sets are not shown.

Table 2: **Large models as executors**: search success (%) by question category without skills and with skills from the learned 4B proposer. Parentheses give changes from execution without skills.

Executor	Skills	Direct retr.	Comparison	Overall
Qwen3.5-397B-A17B	No skills	46.0	49.0	48.0
	SKILLREIN	47.0 (+1.0)	53.0 (+4.0)	51.0 (+3.0)
DeepSeek V4 Pro	No skills	42.0	52.5	49.0
	SKILLREIN	46.0 (+4.0)	55.0 (+2.5)	52.0 (+3.0)

also measures utility beyond training. Recomputed for 384 base and learned skill sets on the batches of Table 1, it ranks proposals consistently with their success changes, with a Spearman correlation of 0.78, and the learned skill sets occupy the high-reward region (Figure 4c).

Large models as executors. We next ask whether the 4B proposer can guide executors that are far larger than itself. Qwen3.5-397B-A17B and DeepSeek V4 Pro replace the Qwen3 executor; each collects its own baseline experience on direct-retrieval and comparison questions, receives one skill set from the learned proposer, and attempts the same questions again (Table 2; Appendix D). The learned skills raise the success of Qwen3.5-397B-A17B from 48.0% to 51.0%, by 1.0 points on direct retrieval and 4.0 points on comparison, and higher than that of DeepSeek V4 Pro from 49.0% to 52.0%. Guidance learned through a small executor can thus improve larger executors.

Robustness across generations. Table 1 reports the first generation of each proposer, so we repeat the untrained and learned proposers with two further generation seeds on all 64 batches (Table 3). Every generation of the learned proposer exceeds every generation of the untrained proposer by at least 14.9 points, and the learned proposer varies by less than one point across seeds, from 35.8% to 36.5%. The learned proposer also produces a valid skill list for every batch, whereas in the first generation the untrained proposer fails on 4 of the 64 batches (Appendix D). The gain from RL is therefore a stable property of the learned policy rather than of a single sample.

4 Related Work

Self-improving agents. Self-improvement can operate on an agent's context, parameters, or implementation. Reflexion (Shinn et al., 2023) uses verbal feedback to inform subsequent attempts, while the Darwin Gödel Machine (Zhang et al., 2026a) searches over modifications to an agent's code. SkillRL (Xia et al., 2026) couples an evolving skill library with reinforcement learning of the task agent. SKILLREIN targets the transformation from experience to reusable guidance. Its sepa-

Table 3: **Repeated generations** with the Qwen3-4B executor: search success (%) of the untrained and learned proposers for each generation seed on the batches of Table 1.

Proposer	Seed 42	Seed 43	Seed 44	Mean
Base Qwen3-4B	20.9	18.8	18.3	19.3
SKILLREIN	36.5	35.8	36.5	36.3

rately trained proposer adapts the context of a fixed executor, making the skill-generation mechanism directly optimizable through downstream execution.

Reusable skills and experience memory. Voyager (Wang et al., 2024) accumulates executable skills for subsequent tasks. ExpeL (Zhao et al., 2024) extracts knowledge from agent experience, Agent Workflow Memory (Wang et al., 2025) induces reusable workflows, and Agentic Context Engineering (Zhang et al., 2026b) organizes experience into evolving contextual playbooks. These systems demonstrate the value of expressing procedural knowledge explicitly and retrieving or revising it as context. Their induction procedures rely on prompted language models whose parameters are not optimized for the utility of the resulting guidance. SKILLREIN learns this transformation through a fixed executor's outcomes, and our experience-memory baseline retains the selected experience directly to isolate the benefit of learning procedures from it.

Learning feedback and skill generators. Retroformer (Yao et al., 2024) trains a retrospective language model with policy gradients to improve a frozen actor through feedback on subsequent attempts. More recent methods train procedural guidance directly. Skill-R1 (Vishe et al., 2026) learns recurrent, instance-level skill revision; Skill-α (Shen et al., 2026) learns progressive edits for a fixed worker with rollback rewards on an anchor query; SAPO (Zhang et al., 2026c) evaluates candidate skills against retrieved context and co-evolves skill generation with the task policy, including induction from grouped queries. Prompt optimization searches over the instructions given to a fixed model through model-written critiques or evolutionary proposals (Yang et al., 2024; Yuksekgonul et al., 2025; Agrawal et al., 2026), and RL teachers learn to write explanations that make known solutions more likely under a student model (Cetin et al., 2025). Our method scores a complete shared skill set through paired outcome differences on tasks reporting each skill, together with coverage of the set. One proposal serves multiple executions, the executor remains fixed, and the trained proposer maps new experience to guidance without any further search.

5 CONCLUSION

We introduced SKILLREIN, an RL framework that learns to generate shared natural-language skills for a frozen executor. The proposer reads the executor's successful and failed trajectories, writes one skill set for a batch of related tasks, and is rewarded by the paired outcome changes on the tasks that cite each skill, together with coverage of the set. On search, the learned proposer produces more useful guidance for the executor it is trained with than its initialization and five much larger models, improves every question category, and lets the executor finish within its action limit with fewer searches and tokens. The gain is stable across generations and carries over to a larger executor that the proposer never saw during training. On WebShop, learning likewise turns an ineffective initialization into a proposer whose skills raise test success several fold. The mechanism that produces guidance can therefore itself be learned from downstream use. Because the training signal consists only of the executor's trajectories and pass rates, the same loop can continue as the executor grows stronger or the environment changes, and training one proposer across several executors and environments would extend the learned experience-to-skill mapping further. Feeding skill-conditioned experience into subsequent proposal rounds is a natural next step toward recursive improvement.

REPRODUCIBILITY STATEMENT

Algorithm 1 and Section 2 specify the proposal, execution, and reward pipeline. Appendix A provides the training configuration, Appendix B describes the skill interface, and Appendix D specifies

the evaluation data and decoding settings. Additional search results and the WebShop setup are provided in Appendices E and F.

AI USE STATEMENT

Generative AI tools assisted with literature discovery and summarization, research framing, experimental design and implementation, result analysis, manuscript drafting and editing, and preparation of the method illustration. The reported numerical results derive from recorded experiment outputs. The authors are responsible for the final text, scientific claims, code, and visual artifacts.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Edoardo Cetin, Tianyu Zhao, and Yujin Tang. Reinforcement learning teachers of test time scaling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 107533–107567, 2025.

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.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O. 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.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural Questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822, 2023.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Junhao Shen, Zhanqiu Zhang, Yiwen Guo, and Hong Cheng. Progressive agent skill generation via reinforcement learning. *arXiv preprint arXiv:2608.01678*, 2026.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Yash Vishe, Rohan Surana, Xunyi Jiang, Zihan Huang, Xintong Li, Nikki Lijing Kuang, Tong Yu, Ryan A. Rossi, Jingbo Shang, Julian McAuley, and Junda Wu. Skill-R1: Agent skill evolution via reinforcement learning. *arXiv preprint arXiv:2605.09359*, 2026.

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*, 2024.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, pp. 63897–63911. PMLR, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. 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.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

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(8055):609–616, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026a.

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 *The Fourteenth International Conference on Learning Representations*, 2026b.

Zhiwei Zhang, Yudi Lin, Nikki Lijing Kuang, Linlin Wu, Xiaomin Li, Songtao Liu, and Fenglong Ma. Co-evolving skill generation and policy optimization. *arXiv preprint arXiv:2606.08755*, 2026c.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Training Configuration

Table 4 lists the training settings of the search proposer, recorded in the configuration of the training run. The proposer starts from Qwen3-4B and is optimized directly by RL with the GiGPO advantage estimator (Feng et al., 2025) and clipped policy updates, while the task executor is a separate, frozen model. Each rollout batch contains 16 task groups with eight candidate proposals each, giving 128 proposal episodes, and every candidate is evaluated on its batch of 50 questions. Candidates of a group share the task batch and are compared through group-relative advantages. Each episode contains a single proposal, so no temporal credit assignment across proposer decisions is required.

Table 4: Training configuration of the search proposer.

Parameter	Value
Proposer initialization	Qwen3-4B
Additional supervised fine-tuning	None
Parameter update	Full parameters; LoRA rank 0
Distributed actor	FSDP; 1 node, 8 GPUs
Gradient checkpointing	Enabled
Advantage estimator	GiGPO
GiGPO step-advantage weight	1.0
GiGPO normalization	Group mean and standard deviation
Similarity-based grouping	Disabled
Discount factor γ / GAE parameter	0.95 / 1.0
Learning rate / weight decay	10^{-6} / 0.01
Warmup ratio / configured schedule steps	0.01 / 1,600
PPO clipping / gradient clipping	0.2 / 1.0
PPO epochs / configured minibatch size	1 / 256
PPO microbatch per GPU	4
Actor KL loss / coefficient	Enabled / 0.001
KL added to environment reward	No
Invalid-action penalty option / coefficient	Enabled / 0.01
Training groups / validation groups	16 / 16
Candidate proposals per task group	8
Questions per environment batch	50
Input / output token limits	8,192 / 2,048
Training temperature / top-p / top-k	1.0 / 1.0 / unrestricted
Built-in validation decoding	Greedy, temperature 0
Outer proposal steps / history	1 / 0
Inner executor actions / history	4 / 4
Search results per retrieval	3
Skill-use reward weight λ	0.05
Reward scale c	1.0 (environment default)
Environment seed	42
Configured epochs	100
Checkpoint / validation interval	Every 4 training steps

B Skill Interface and Execution Feedback

Experience input. The proposer prompt contains the task category and its description, the baseline success rate, the executor's action budget, and selected execution records. The formatter takes up to four successes and six failures in their original order and does not fill an unavailable quota with records of the other outcome. Each record keeps the question and its outcome label, and for each step the first 300 characters of the action, with newlines replaced by spaces, and the first 180 characters of the observation. The prompt states that each of the four available actions is either a search or a final answer. All proposers receive the same input for a given executor and batch.

Skill output. Each skill contains `skill_id`, `title`, `principle`, and `when_to_apply`, and the prompt requests about five skills. The following entry was generated by the learned proposer for the Qwen3-4B executor:

ID: `qts_002`
Title: Step Necessity Check
Principle: Avoid redundant search steps. If current evidence fully satisfies the question's requirements, bypass unnecessary searches and answer directly.
When to apply: After each search, evaluate whether the retrieved info bridges the gap to the final answer; if so, proceed to answer.

The parser extracts the skill list from the response and requires nonempty titles and principles. A nonempty list is inserted with the skill-use wrapper, which begins with "MANDATORY OPERATING PROCEDURE", asks the executor to consult the skills before every action, to name the identifier and relevant fragment of the skill it follows, and to report the applied identifiers after each search or answer. Execution without skills, experience memory, and empty skill lists run without the wrapper, so comparisons among proposers hold the wrapper fixed. Unparsable generations remain in the results and run without skills.

Execution feedback. The evaluator stores actions, retrieved observations, answer correctness, and reported skill identifiers. Identifiers in the `<used_skills>` field determine the use sets in Eq. (5). The guided trajectories supply the reward for the parameter update, and the next episode begins with baseline execution on a newly sampled batch.

C Reward Aggregation and Cost Accounting

For $m > 0$, let A be the subset of proposed skills cited at least once and let $A_{\text{align}} \subseteq A$ contain those with at least one reported application that can be matched to a baseline task. The implementation computes $q = |A|/m$ and averages ℓ_j over A_{align}. With complete paired batches, $A_{\text{align}} = A$, recovering Eq. (5). An empty aligned set gives $L = 0$; a nonempty proposal with no reported use therefore receives $-c\lambda$, while a cited proposal can receive a larger penalty if outcomes regress. If the parser produces no skill, the proposal receives $-c(1 + \lambda)$.

Equation (7) in the main text gives the equivalent task weighting. Since $y_i^1 - y_i^0 \in [-1, 1]$ and $q \in [0, 1]$, the reward lies in $[-c(1 + \lambda), c(1 + \lambda)]$ for $c \geq 0$. The outer environment passes Eq. (6) to the trainer, replacing the execution service's aggregate success-difference score. The configured invalid-action penalty and KL term belong to the training stack and are listed separately in Table 4.

For evaluation, we distinguish baseline collection C_0, skill generation G, and guided execution C_1. A complete adaptation episode has cost

$$C_{\text{episode}} = C_0 + G + C_1. \tag{8}$$

Figure 2 reports C_1 per attempted task, averaging over successes and failures, with searches and executor tokens, input plus output, accumulated until the trajectory ends. Skill generation G is incurred once per batch and shared by its 50 tasks.

D Evaluation Protocol

Evaluation batches. Questions from the seven datasets are routed to the four categories by the original classifier. The evaluation samples 64 batches of 50 questions, 16 per category, from the training question pool with fixed seeds, giving 3,200 question occurrences and 2,617 distinct question texts. Success averages all occurrences, including questions repeated across batches. Within an executor and batch, every proposer receives the same baseline success rate and selected records, and its skill set is applied to all 50 questions, including the selected examples. The evaluation therefore measures adaptation to a batch after observing its labeled baseline experience.

Checkpoints and runs. Table 1 reports the first generation of every method, without selection by quality, using the released `search_200` checkpoint and fresh baseline experience from each executor, and all other analyses on these batches use the same checkpoint. The Qwen3-8B results in Figure 2a and Table 6 come from an earlier run on the same batches, in which the learned proposer was served from the training run and compared with GLM-5.2, DeepSeek V4 Pro, and Qwen3.5-397B-A17B. The large executors of Table 2 each answer 100 direct-retrieval and 200 comparison questions from these batches. Every learned proposer in this paper has the Qwen3-4B architecture, and Qwen3-8B serves only as a receiving executor.

Execution environment. The search service uses the Wiki18 corpus of 21,015,324 passages, E5-base-v2 embeddings (Wang et al., 2022), an FP16 flat index, and three retrieved passages per search, together with the original category routing, action parser, four-action budget, and exact-match answer evaluator. The Qwen3 executors decode greedily in BF16 with thinking disabled, at most 1,024 new tokens per action, and a history of four actions. Baseline and guided execution use the same executor, task list, tools, and settings.

Generation settings. The base and learned proposers use thinking, temperature 0.7, top-$p = 0.95$, top-$k = 20$, at most 2,048 output tokens, and generation seeds 42, 43, and 44, of which seed 42 gives the first generation. Opus 5 and GPT-6 Astra run through their official command-line interfaces with medium reasoning, and the API proposers of Table 5 use temperature 0 and a 2,048-token output limit. In the first generation, the base proposer yields valid skill lists for 60 of 64 batches with the experience of the Qwen3-4B executor and 52 of 64 with that of the Qwen3-8B executor, and the learned proposer yields valid lists for all batches.

Table 5: API identifiers of the large models.

Model	API identifier
GLM-5.2	`z-ai/glm-5.2`
DeepSeek V4 Pro	`deepseek/deepseek-v4-pro`
Qwen3.5-397B-A17B	`qwen/qwen3.5-397b-a17b`

E Additional Search Results

Qwen3-8B executor. Table 6 gives the category breakdown behind Figure 2a for the Qwen3-8B executor, which was not used in training. Every proposer improves over execution without skills, and the learned proposer gives the highest overall success and the best result in three of the four categories. When the skill sets are regenerated with the released checkpoint, the learned proposer raises the success of this executor from 18.3% to 29.2%, while GLM-5.2, DeepSeek V4 Pro, and Qwen3.5-397B-A17B reach 31.4% to 34.5%.

Table 6: **Qwen3-8B executor**: search success (%) by question category on the batches of Table 1. Parentheses give changes from execution without skills, and bold marks the best proposer in each column.

Method	Direct retr.	Multi-hop	Entity attr.	Comparison	Overall
No skills	19.4	21.5	14.7	19.0	18.6
GLM-5.2	27.9 (+8.5)	29.9 (+8.4)	21.4 (+6.7)	26.0 (+7.0)	26.3 (+7.7)
DeepSeek V4 Pro	**33.6** (+14.2)	28.3 (+6.8)	25.0 (+10.3)	22.9 (+3.9)	27.4 (+8.8)
Qwen3.5-397B-A17B	26.5 (+7.1)	26.9 (+5.4)	24.1 (+9.4)	28.1 (+9.1)	26.4 (+7.8)
SkillRein	28.3 (+8.9)	**34.9** (+13.4)	**25.2** (+10.5)	**32.1** (+13.1)	**30.1** (+11.5)

F WebShop

Training. The WebShop proposer is a separate Qwen3-4B model trained with the outcome-and-use reward through a frozen Qwen3-4B executor that decodes without thinking and acts for at most ten steps. Training uses batches of 50 synthetic instructions over a development catalog of 1,000 products, eight groups of eight proposals per update, learning rate 10^{-6}, no KL penalty, and coverage weight $\lambda = 0.01$. On 400 validation instructions of this development environment, guided success rises from 1.0% at initialization to 13.75% at step 200 and 18.25% at step 260, while the executor alone stays between 1.5% and 2.25%.

Test batches. Figure 3 evaluates the learned proposer in its training environment on one test batch of 50 instructions for each of six product categories. The executor first attempts each batch without

skills, the proposer writes one skill set from this experience, and the executor attempts the batch again with the skills.

G LIMITATIONS

Each episode contains a single proposal followed by guided execution. Recursive rounds, in which guided trajectories feed the next proposal and the proposer keeps learning as the executor improves, would move the approach further toward recursive self-improvement and remain future work. The search proposer is trained through one executor, and training a single proposer across several executors or environments is a natural extension. Skills are delivered as natural-language procedures in the executor's context, and other interfaces, such as executable tools, are not explored. Ablations of individual reward terms are also left to future work.