A^3-SKILL: LEARNING AUTONOMOUS MANAGEMENT OF EXECUTABLE TOOL SKILLS VIA PURE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) increasingly rely on external tools to solve complex tasks. Tool skills enable LLMs to reuse procedural knowledge from past tool interactions. Existing approaches, however, often rely on fragmented skill learning, skill-operation supervision, or natural-language skill representations. We study autonomous tool-skill management: jointly acquiring, applying, and adapting executable tool skills within a single policy. We propose A^3-Skill, a two-stage reinforcement learning framework without supervised skill operations. In the first stage, we adopt a progressive curriculum that gradually transitions from reference-guided skill construction to autonomous skill acquisition and refinement. In the second stage, a leave-one-scenario-out strategy trains the policy to retain, revise, or rebuild imperfectly matched transferred skills through successive task interactions. We evaluate A^3-Skill on AppWorld across independent transfer, sequential adaptation, and cold-start acquisition, and additionally assess cross-benchmark generalization on BFCL-v3. On AppWorld, A^3-Skill improves TGC over the strongest competing baseline by 13.63% under sequential adaptation and 15.78% under cold-start acquisition. It also achieves 32.00% accuracy on BFCL-v3, compared with 28.00% for the strongest baseline.

1 INTRODUCTION

Large language models (LLMs) increasingly rely on external tools to solve complex tasks beyond their inherent capabilities (Qu et al., 2025; Qian et al., 2025). In real-world settings, tasks often require sequential calls of multiple tools and recurring tool-use procedures, including similar tool-call sequences, control flow, and intermediate data dependencies (Chen et al., 2026). For instance, a typical data processing task may repeatedly involve locating candidate records, fetching target data, revising data details, and writing updates back through the corresponding tools. Solving each task independently through atomic tools requires LLMs to repeatedly rediscover similar procedures, leading to redundant reasoning and limited reuse of past experience. To address this issue, recent studies have explored *tool skills*, which encapsulate reusable procedural knowledge from previous tool interactions to facilitate future task solving (Chen et al., 2026; Wang et al., 2026b).

Existing research explores two main approaches to acquiring and reusing tool skills. *Training-free approaches* leverage prompting and test-time interactions to construct, refine, and reuse skills without updating the LLM's parameters (Li et al., 2026b; Wang et al., 2026a). While effective, these approaches rely largely on the LLM's inherent instruction-following capabilities, without explicitly optimizing its behavior. By contrast, *training-based approaches* incorporate skill capabilities into model parameters to improve skill acquisition and utilization (Wang et al., 2026b; Yang et al., 2026a).

Despite these advances, existing approaches still face several limitations. **1) Fragmented skill learning.** Skill construction, execution, and refinement are often learned or handled by separate models, despite their strong interdependence. Optimizing these operations in isolation can lead to globally suboptimal skill-management performance (Zhang et al., 2026; Yang et al., 2026b). **2) Skill-operation supervision.** Bootstrapping skill management with generated skill-operation demonstrations incurs additional supervision costs (Wang et al., 2026b; Zhang et al., 2026).

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.

3) Natural-language skill representations. Many approaches represent skills as textual guidance (Figure 1), requiring LLMs to translate them into concrete tool calls whenever they are used, which can introduce inaccurate and redundant interpretation (Li et al., 2026b; Wang et al., 2026a).

Motivated by these limitations, we study *autonomous tool-skill management*, aiming to enable a tool-using LLM to acquire, apply, and adapt tool skills within a single policy without supervised skill operations. Specifically, each tool skill is an executable program encapsulating a reusable tool-use workflow (Figure 1). Achieving this goal presents three key challenges: **1) Entangled optimization signals.** Since various skill operations can jointly affect downstream task performance, it is difficult to attribute task success or failure to individual skill operations and optimize them accordingly. **2) Limited skill-learning guidance.** Without supervised skill operations, LLMs have to learn skill knowledge indirectly from task interactions, which provide limited guidance for skill learning (Yang et al., 2026a). **3) Strict skill reuse.** Previously learned skills may not fully match new tasks, thus indiscriminate reuse can cause negative transfer (Huang et al., 2026; Dong et al., 2026). Executable tool skills are particularly sensitive to such mismatch, since even small discrepancies in tools or workflow structure can invalidate their reuse.

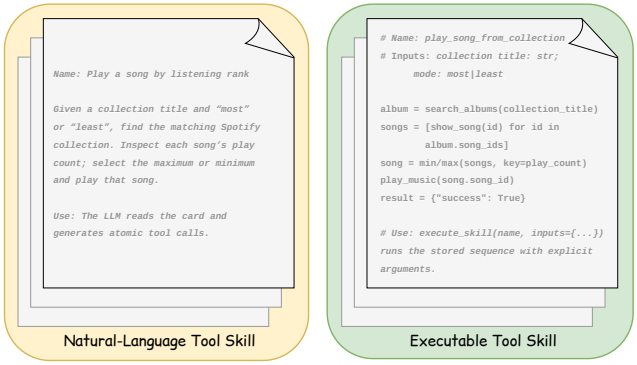


Figure 1: Natural-language skills require tool-call generation at each reuse; executable skills store a parameterized workflow.

To address these challenges, we propose **A^3-Skill**, a two-stage reinforcement learning (RL) framework that enables a single LLM to progressively acquire executable tool skills and further adapt them across tasks. To address the first two challenges, the first stage progressively develops unified skill-management capabilities within a single LLM. We employ a progressive curriculum that starts with successful tool-use experience to guide skill construction and gradually transitions to the LLM's own interactions for autonomous skill construction, execution, and refinement. This curriculum progressively increases the participation of different skill operations, easing optimization before full joint learning. To provide more skill-learning guidance, we evaluate constructed skills using both downstream task success and tool-use workflow alignment, explicitly rewarding whether a skill captures reusable procedural structure. To address the third challenge, the second stage deliberately exposes the LLM to imperfectly matched skills to enhance skill adaptation. We adopt a *leave-one-scenario-out* strategy that requires the LLM to adapt previously learned skills from other task scenarios to the target scenario, where it can learn to retain, revise, or rebuild them through successive task interactions.

We evaluate A^3-Skill on AppWorld (Trivedi et al., 2024) across independent transfer, sequential adaptation, and cold-start acquisition, assessing skill application, adaptation, and acquisition, respectively. A^3-Skill achieves the strongest average performance, with relative TGC/SGC gains over the strongest competing baseline of 13.63%/22.22% under sequential adaptation and 15.78%/28.56% under cold-start acquisition. On BFCL-v3 (Patil et al., 2025), it achieves 32.00% accuracy versus 28.00% for the strongest baseline, demonstrating superior skill adaptation or acquisition capability.

2 Related Work

Tool learning. Tool learning extends LLMs beyond their parametric knowledge through external tools (Qu et al., 2025). Training-free approaches use prompting and interaction feedback, while training-based methods optimize tool selection, argument generation, and multi-step execution through supervised learning or reinforcement learning (Schick et al., 2023; Patil et al., 2024; Qian et al., 2025). Recent agentic RL methods optimize long-horizon behavior against environment

outcomes (Zhang et al., 2025). Whereas these methods primarily improve individual task execution, we study how recurring tool-use experience can be acquired and adapted as reusable skills across tasks.

Agent skills. Agent skills encode reusable procedural knowledge as natural-language guidance, executable programs, or structured workflows (Zhou et al., 2026). Training-free methods construct and refine skills through prompting and test-time interactions (Chen et al., 2026; Li et al., 2026b; Wang et al., 2026a). Training-based methods learn skill generation, utilization, or maintenance through model optimization (Wang et al., 2026b; Xia et al., 2026; Li et al., 2026a; Yang et al., 2026a). Some separate skill construction from the acting policy (Zhang et al., 2026; Yang et al., 2026b), and transferred skills can become ineffective on mismatched tasks (Huang et al., 2026; Dong et al., 2026). Our focus is unified acquisition, application, and adaptation of executable tool skills through RL, without supervised skill operations. Appendix A compares representative methods along these three design dimensions.

3 METHODOLOGY

We propose **A^3-Skill**, a two-stage framework that enables a single LLM to learn skill *acquisition*, *application*, and *adaptation* through reinforcement learning. Figure 2 summarizes the framework. The LLM represents reusable tool-use procedures as executable programs and manages them through a shared skill library. In Stage I, the LLM progressively transitions from constructing skills with successful tool-use trajectories to acquiring them entirely from its own interactions. In Stage II, the LLM is provided with skills learned from other scenarios and learns to keep, update, or build them in the target scenario. Policy optimization uses only interaction rewards, without supervised skill operations; early reference trajectories serve as context rather than SFT targets.

3.1 TASK FORMULATION

We consider a tool-use task $\tau = (x, \mathcal{T}, \mathcal{E})$, where x is a natural-language instruction, $\mathcal{T}$ is the set of available tools, and $\mathcal{E}$ is an interactive environment. At each turn t, the LLM policy π_θ generates a program action a_t based on the current interaction history. Executing the action produces an observation o_{t+1}, resulting in a tool-use trajectory $\xi_\tau = (o_1, a_1, o_2, a_2, \ldots, o_T, a_T, o_{T+1})$. After the LLM submits a completion, the environment returns a binary task outcome $y_\tau \in \{0, 1\}$.

Executable tool skills. We represent an executable tool skill as $s = (P_s, C_s, G_s)$, where P_s is an executable workflow program, C_s is its reuse contract, and G_s is its workflow graph. The reuse contract contains a short description, typed inputs and outputs, and the conditions under which the skill can be applied. The workflow graph $G_s = (V_s, E_s)$ summarizes the program structure: nodes in V_s denote tool operations, while edges in E_s represent their execution order or data dependencies. Unlike natural-language guidance, P_s can be directly executed without regenerating its internal tool calls.

Skill-management operations. The LLM interacts with a skill library $\mathcal{M}$ through four operations:

$$\mathcal{A}_{\text{skill}} = \{\text{EXECUTE}, \text{BUILD}, \text{UPDATE}, \text{KEEP}\}. \tag{1}$$

EXECUTE applies an existing skill with task-derived inputs without modifying the library. BUILD adds a new executable skill. UPDATE creates a revised version of an existing skill while preserving its lineage. KEEP retains the current library without modification.

3.2 STAGE I: PROGRESSIVE SKILL ACQUISITION

To facilitate skill reuse across related tasks, we group tasks with similar user instructions and tool-use workflows into scenarios. For each task, we collect a tool-use trajectory that solves the task as a successful reference trajectory. During training, we sample three distinct task variants τ^a, τ^b, and τ^c from each scenario.

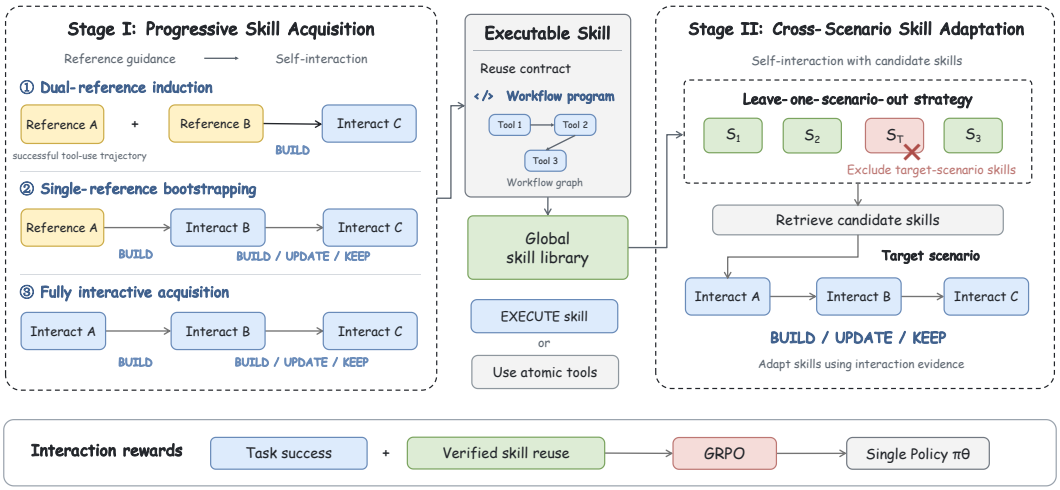


Figure 2: Overview of A^3-Skill. Stage I progressively transitions from reference-guided skill construction to fully interactive skill acquisition, while Stage II learns cross-scenario skill reuse and adaptation through a leave-one-scenario-out strategy. Both stages optimize the same tool-using policy with task-success and verified skill-reuse rewards.

Progressive curriculum. Learning to construct executable skills directly from sparse task outcomes is difficult. We therefore introduce a curriculum with three acquisition modes.

In *dual-reference induction*, the LLM observes successful reference trajectories for τ^a and τ^b. It compares their tool-use procedures, identifies shared operations, parameterizes task-specific values, and invokes BUILD to construct an executable skill. The skill is subsequently evaluated on the held-out task τ^c.

In *single-reference bootstrapping*, the LLM first constructs a skill from a successful reference trajectory for τ^a. It then solves τ^b through interaction and uses the resulting trajectory to retain, revise, or rebuild the skill before applying it to τ^c.

In *fully interactive acquisition*, no reference trajectory is provided. The LLM solves τ^a using the available tools and consolidates the resulting experience into its local skill library. It repeats this process on τ^b and finally applies the resulting skill library to τ^c. The full interaction sequence is

$$\tau^a \to d^a \to \tau^b \to d^b \to \tau^c, \qquad d^a, d^b \in \{\text{BUILD}, \text{UPDATE}, \text{KEEP}\}. \tag{2}$$

At training step k, an acquisition mode is sampled according to

$$z_k \sim \text{Cat}\left(w_k^{\text{dual}}, w_k^{\text{single}}, w_k^{\text{online}}\right). \tag{3}$$

The curriculum gradually decreases w_k^{dual} and increases w_k^{online}, gradually shifting skill acquisition from successful reference trajectories toward the LLM's own interaction experience.

Task-success and skill-quality rewards. A successful skill construction alone does not guarantee that the skill is useful. We therefore verify each candidate through its downstream use. For a skill s and held-out task τ^c, we define its verified reuse utility as

$$U(s, \tau^c) = y_{\tau^c} \cdot \mathbb{I}[s \text{ is executed and used}] \cdot S_{\text{wf}}(G_s, G_c), \tag{4}$$

where G_c is the workflow graph observed on τ^c. Since a skill may cover only part of a task, we compare it with the best-matching subgraph of G_c:

$$S_{\text{wf}}(G_s, G_c) = \max_{H \subseteq G_c} \left[F_1(V_s, V_H) + F_1(E_s, E_H)\right]. \tag{5}$$

Here H is a subgraph of the observed workflow; V_s and V_H contain tool operations, while E_s and E_H contain their execution-order and data-dependency relations. For either type of graph element,

F_1 measures the overlap between the two sets. Thus, $U(s, \tau^c)$ rewards a skill only when it contributes to a successful task and its workflow matches a relevant part of the observed tool use.

The construction and held-out execution rewards are

$$R_{\text{build}} = U(s, \tau^c) - N_{\text{build}}, \tag{6}$$

$$R_{\text{exec}} = y_{\tau^c} + U(s, \tau^c) - N_{\text{exec}}, \tag{7}$$

where N_{build} and N_{exec} count invalid construction and execution actions. The delayed utility term in R_{build} assigns credit to the construction action that produced a skill later validated on τ^c.

After training, we retain skills with positive held-out reuse utility. When multiple candidates originate from the same scenario, we keep the one with the highest empirical execution reliability. The resulting verified global library is denoted by $\mathcal{M}^{(1)}$.

3.3 Stage II: Cross-Scenario Skill Application and Adaptation

Stage II teaches the LLM to reuse skills beyond the scenarios from which they were acquired. For a target scenario σ, we construct a leave-one-scenario-out library

$$\mathcal{M}_{-\sigma} = \{s \in \mathcal{M}^{(1)} \mid \text{src}(s) \neq \sigma\}, \tag{8}$$

where $\text{src}(s)$ denotes the source scenario of skill s. This exclusion prevents the LLM from retrieving a skill constructed from another variant of the target scenario.

Semantic retrieval. For a task τ in target scenario σ, we retrieve candidate skills from the leave-one-scenario-out library $\mathcal{M}_{-\sigma}$. We compare the task instruction x with each skill's reuse contract C_s:

$$S_{\text{sem}}(\tau, s) = \cos(e(x), e(C_s)), \tag{9}$$

where $e(\cdot)$ is a text encoder and $\cos(\cdot, \cdot)$ denotes cosine similarity. We select the K skills with the highest scores:

$$\mathcal{R}_K(\tau) = \text{TopK}_{s \in \mathcal{M}_{-\sigma}} S_{\text{sem}}(\tau, s), \tag{10}$$

where K is the retrieval budget. The LLM receives the selected skills' reuse contracts and executable interfaces, then decides which are suitable for the task.

Selective application. The retrieved skills are candidates rather than mandatory actions. Before execution, the LLM compares each skill's contract with the task instruction, including its target object, operation, filtering conditions, and semantic direction. If a skill is compatible, the LLM instantiates and executes it using task-derived inputs. Otherwise, it solves the task through atomic tools. Thus, rejecting an incompatible skill is treated as valid behavior rather than failed reuse.

Evidence-based adaptation. For each target scenario, the LLM processes three task variants sequentially:

$$\tau^a \to d^a \to \tau^b \to d^b \to \tau^c, \qquad d^a, d^b \in \{\text{BUILD}, \text{UPDATE}, \text{KEEP}\}. \tag{11}$$

After each previous task attempt, the LLM examines the generated program, tool outcomes, skill calls, and task result, then selects one of the previously defined KEEP, UPDATE, or BUILD operations. An update addresses a reusable defect supported by the execution evidence; a new skill is built only from a successful tool-use trajectory.

Adaptation rewards. Let y_i be the task outcome for target task τ^i. We use U_i^{ret} to denote the verified utility of a retrieved skill on task τ^i, defined by successful task completion, actual skill use, and workflow-graph similarity. Similarly, $U_{j \to i}^{\text{build}}$ and $U_{j \to i}^{\text{update}}$ indicate that a skill built or revised after task τ^j is successfully reused on a later task τ^i. The Stage II episode reward is

$$R_{\text{II}} = \sum_i y_i + \sum_i U_i^{\text{ret}} + \sum_{j<i} U_{j \to i}^{\text{build}} + \sum_{j<i} U_{j \to i}^{\text{update}} - N_{\text{err}}, \tag{12}$$

where N_{err} counts invalid skill executions and tool actions. When a newly built or revised skill benefits a later task, its utility reward is assigned to the earlier phase that produced the corresponding library decision. This delayed credit encourages changes that improve subsequent tasks rather than changes that merely execute successfully in isolation.

3.4 PURE REINFORCEMENT LEARNING

The same LLM policy π_θ is optimized in both stages using group relative policy optimization (GRPO) (Shao et al., 2024). For a group of B rollouts with returns $\{R_j\}_{j=1}^{B}$, we compute

$$\widehat{A}_j = \frac{R_j - \mathrm{mean}(\{R_k\}_{k=1}^{B})}{\mathrm{std}(\{R_k\}_{k=1}^{B}) + \epsilon}. \tag{13}$$

The policy objective is

$$\mathcal{L}_{\mathrm{GRPO}} = -\mathbb{E}_{j,t}\left[\min\left(\rho_{j,t}\widehat{A}_j,\ \mathrm{clip}(\rho_{j,t}, 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\widehat{A}_j\right)\right], \tag{14}$$

where $\rho_{j,t}$ is the probability ratio between the current and rollout policies.

4 EXPERIMENTS

We evaluate A^3-Skill to answer three questions: (1) whether autonomous skill management improves tool-use performance beyond direct execution and task-only reinforcement learning; (2) whether the learned policy can acquire and adapt reusable skills through successive interactions; and (3) which components are necessary for learning effective executable skill management without supervised skill-management demonstrations.

4.1 EXPERIMENTAL SETUP

Benchmark. Our main evaluation uses the test-normal split of AppWorld (Trivedi et al., 2024), containing 168 tasks from 56 scenarios, with three task variants per scenario. We report Task Goal Completion (TGC), the fraction of completed tasks, and Scenario Goal Completion (SGC), the fraction of scenarios with all three tasks completed. We additionally evaluate cross-benchmark generalization on BFCL-v3 (Patil et al., 2025); the protocol is described in Appendix B.3, with results in Section 4.5.

Evaluation settings. We consider three settings with progressively stronger requirements on skill management. In **Independent Transfer**, each task independently retrieves candidates from the frozen global training library, without updating the library. In **Sequential Adaptation**, a candidate set is retrieved before the first task of each scenario, and the model processes three variants sequentially as $A \to B \to C$, making local BUILD, UPDATE, or KEEP decisions after A and B. In **Cold-start Acquisition**, each scenario starts from an empty library and the model must acquire and refine reusable skills solely through the $A \to B \to C$ interactions. The three settings therefore primarily assess skill application, adaptation, and autonomous acquisition, respectively.

Baselines. For controlled comparison, all baselines use Qwen3-4B-Instruct-2507 as the acting model and share the same tool interface and evaluation budget. We adapt the key training or deployment paradigm of each prior method to our executable-skill environment. **Direct** directly solves tasks with atomic tools and maintains no skill state. **ToolRL** (Qian et al., 2025) is a task-only RL baseline trained with outcome-based GRPO on the same tool-use tasks, without skill-management actions. **SkillCraft** (Chen et al., 2026) is a prompting-only skill-use baseline in which the frozen Qwen3-4B policy is provided with the executable global library learned by A^3-Skill. **MIND-Skill** (Li et al., 2026b) represents an expert-generated-library baseline: a stronger teacher constructs and verifies executable skills offline, which are subsequently used by the frozen Qwen3-4B policy. **SAGE** (Wang et al., 2026b) represents expert-bootstrapped skill learning, where expert-generated skill-management trajectories are first used for SFT before reinforcement learning. These adaptations allow us to compare task-only RL, prompted skill use, expert-built skills, supervised skill learning, and our RL-only autonomous skill learning under a common acting model.

4.2 MAIN RESULTS

Overall performance. Table 1 presents the main results across the three evaluation settings. Direct and ToolRL maintain no skill library or cross-task state and therefore exhibit the same behavior

Methods	INDEPENDENT		SEQUENTIAL		COLD-START		ALL	
	TGC	SGC	TGC	SGC	TGC	SGC	TGC	SGC
Direct	12.50	5.36	12.50	5.36	12.50	5.36	12.50	5.36
ToolRL	14.29	7.14	14.29	7.14	14.29	7.14	14.29	7.14
SkillCraft	17.86	10.71	16.07	7.14	14.88	7.14	16.27	8.33
MIND-Skill	18.45	10.71	15.48	7.14	–	–	16.97†	8.93†
SAGE	**25.00**	**14.29**	26.19	16.07	22.62	12.50	24.60	14.29
A^3-Skill	24.40	**14.29**	**29.76**	**19.64**	**26.19**	**16.07**	**26.78**	**16.67**
% improve	*-2.40%*	*0.00%*	*13.63%*	*22.22%*	*15.78%*	*28.56%*	*8.86%*	*16.66%*

Table 1: Main results (%) on AppWorld test-normal. **Bold** denotes the best value (including ties); underlining denotes the second-best distinct value. The improvement row compares A^3-Skill with the strongest competing baseline. ALL averages the three settings. †MIND-Skill averages only the two settings in which an expert-generated initial library is applicable.

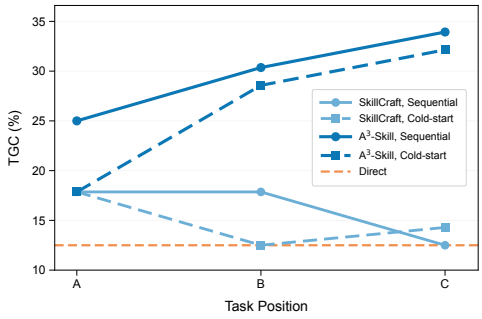


Figure 3: TGC across successive tasks within each AppWorld scenario.

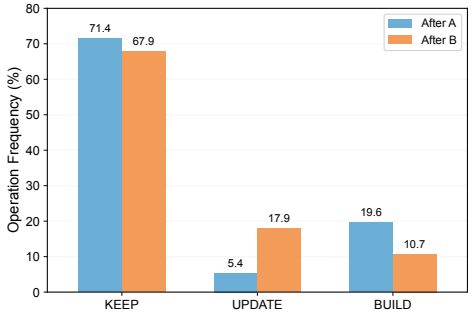


Figure 4: Distribution of skill-management decisions after successive tasks.

across the three protocols. Task-only RL provides only modest improvement over Direct, suggesting that A^3-Skill's gain cannot be explained by reinforcement learning alone. The expert-generated MIND-Skill library improves independent reuse but does not itself teach the frozen policy how to adapt transferred skills. A^3-Skill achieves the strongest average performance, reaching 26.78 TGC and 16.67 SGC across all settings. SAGE slightly outperforms A^3-Skill in Independent Transfer TGC (25.00 vs. 24.40), suggesting that expert-supervised initialization can be advantageous when the primary requirement is direct reuse of an existing skill. However, A^3-Skill shows substantially larger gains once interaction requires skill adaptation or autonomous acquisition. Under Sequential Adaptation, it reaches 29.76 TGC and 19.64 SGC, outperforming SAGE by 13.63% and 22.22% relatively. The advantage further persists under Cold-start Acquisition, where no reusable skill is initially available. These results suggest that the main benefit of A^3-Skill lies in learning to adapt and acquire skills through interaction rather than merely exploiting a well-initialized skill library.

Performance across successive task interactions. Aggregate performance alone does not reveal whether the policy benefits from experience accumulated within a scenario. We therefore decompose TGC according to task position A, B, and C in Figure 3. The prompting-only SkillCraft policy does not consistently improve across successive tasks. In contrast, A^3-Skill exhibits a clear upward trend under both Sequential Adaptation and Cold-start Acquisition. The increase is particularly pronounced under cold start, where no reusable skill is initially available, providing direct evidence that the policy acquires and improves reusable skills from its own preceding interactions.

4.3 ABLATION STUDY

Removing Stage II primarily degrades Sequential Adaptation, confirming the importance of explicitly learning to adapt transferred skills across scenarios. Removing the progressive curriculum affects both settings and causes a larger degradation under cold start, where skill acquisition must

Table 2: Ablation TGC (%): Sequential Adaptation (Seq.) and Cold-start Acquisition (Cold).

Ablation Variants	Seq. TGC	Cold TGC
A^3-Skill	**29.76**	**26.19**
w/o Stage II	26.79	25.00
w/o Curriculum	22.62	19.64
w/o UPDATE	27.38	24.40
w/o BUILD	25.60	20.83

Table 3: Skill utilization and successful reuse (%) under Sequential Adaptation.

Methods	Skill Utilization	Successful Reuse
SkillCraft	5.36	0.00
A^3-Skill	**27.97**	**21.05**

be learned from sparse interaction feedback. The two lifecycle operations show complementary effects: removing UPDATE moderately hurts adaptation because partially compatible skills can still be rebuilt, whereas removing BUILD has only a limited effect when a global library is available but substantially degrades cold-start performance. Importantly, none of these variants collapses to the base-model level, since the policy retains its learned direct tool-use capability even when a particular skill-management operation is unavailable.

Reward ablations appear in Appendix C.1; additional skill analyses appear in Appendix C.2.

4.4 IN-DEPTH ANALYSIS

Analysis on skill reuse. We first examine whether reinforcement learning changes how the policy actually uses retrieved skills. Skill Utilization measures the fraction of retrieved candidates that are executed, while Successful Reuse measures how often skill execution contributes to successful task completion. As shown in Table 3, SkillCraft rarely invokes the retrieved skills despite having access to exactly the same global library. A^3-Skill substantially increases both utilization and successful reuse, suggesting that RL improves the use of retrieved knowledge. Retrieval itself uses a fixed text encoder.

Analysis on skill-management decisions. We next analyze how the learned policy changes its local skill state as more execution evidence becomes available. Figure 4 shows the distribution of KEEP, UPDATE, and BUILD decisions after tasks A and B. Most interactions preserve the current library, indicating that the policy does not modify skills indiscriminately. More importantly, BUILD is relatively more frequent after the first task, whereas UPDATE becomes substantially more frequent after the second task. This shift is consistent with first acquiring uncovered workflows and then revising partially compatible skills after observing additional evidence.

Analysis on executable vs. textual skills. A central design choice of A^3-Skill is to represent reusable workflows as executable programs rather than natural-language procedures. To isolate this effect, we construct a textual variant that preserves the same skill-learning and retrieval pipeline but stores each skill as a natural-language procedure that must be grounded into atomic tool calls at every reuse. Table 4 shows that executable skills improve both task completion and generation efficiency. The difference is particularly pronounced in generated code, supporting our hypothesis that executable representations avoid repeatedly translating the same procedural knowledge into low-level tool actions. We do not report Skill Utilization for the textual variant because textual conditioning does not provide an observable skill-execution event comparable to invoking an executable skill.

Methods	TGC ↑	SGC ↑	Code Tokens ↓
A^3-Skill (textual)	26.19	16.07	2180
A^3-Skill	**29.76**	**19.64**	**1325**

Table 4: Comparison of textual and executable skill representations under Sequential Adaptation.

Analysis on execution efficiency. Finally, we examine whether reusable executable skills reduce the cost of solving tool-use tasks. Table 5 reports the average number of generated API call sites, interaction turns, and generated code tokens per task under Sequential Adaptation. Task-only RL does not improve execution efficiency: although ToolRL can improve task success, its trajectories become slightly longer than those of the base model, with more tool interactions and generated code. In contrast, skill-based methods reduce repeated code generation by reusing previously acquired workflows. A^3-Skill achieves the lowest cost across all three metrics, indicating that autonomous skill management improves not only task completion but also the efficiency of solving recurring tool-use workflows.

Methods	**API Call Sites** $\downarrow$	**Turns** $\downarrow$	**Code Tokens** $\downarrow$
Direct	39.80	4.62	3458
ToolRL	41.30	4.86	3860
SkillCraft	34.10	4.02	2320
SAGE	30.30	3.42	1490
A^3-Skill	**27.10**	**3.12**	**1325**

Table 5: Average execution cost per task under Sequential Adaptation. API call sites count generated call expressions, not all API invocations inside stored skills.

4.5 Cross-Benchmark Generalization

We additionally evaluate the BFCL-v3 Base Multi-Turn category using 50 training instances and 150 held-out test instances (Appendix B.3). As shown in Table 6, A^3-Skill achieves 32.00% accuracy, compared with 28.00% for the strongest competing baseline. This supports the applicability of learned skill management beyond AppWorld. Because target-benchmark training instances are available for adaptation, this experiment measures cross-benchmark generalization rather than zero-shot transfer.

Method	Direct	ToolRL	SkillCraft	MIND-Skill	SAGE	**A^3-Skill**
Accuracy	16.00	20.67	23.33	24.67	28.00	**32.00**

Table 6: Accuracy (%) on the held-out BFCL-v3 Base Multi-Turn split.

5 Conclusion

In this work, we introduced A^3-Skill, a reinforcement learning framework for autonomous management of executable tool skills within a single tool-using LLM. Rather than relying on separately optimized skill components or supervised skill-management demonstrations, A^3-Skill jointly learns to acquire, apply, and adapt reusable executable workflows through interaction. A progressive acquisition curriculum facilitates skill learning from increasingly autonomous experience, while leave-one-scenario-out training improves adaptation to imperfectly matched transferred skills. Experiments on AppWorld and BFCL-v3 demonstrate consistent benefits in autonomous skill adaptation and acquisition, together with improved skill reuse and execution efficiency. Future work will explore scaling autonomous tool-skill learning to larger models, longer interaction horizons, and richer skill libraries that support composition and long-term evolution.

Limitations. Our experiments use Qwen3-4B-Instruct-2507, so absolute performance remains bounded by its tool-use and code-generation capabilities. Controlled comparisons nevertheless demonstrate gains from the proposed framework at fixed model capacity. Evaluation with larger models remains future work.

AI USE STATEMENT

We used large language models to assist with manuscript drafting and language polishing, literature retrieval and organization, and research ideation and execution, including feedback on methodological and experimental design as well as assistance with code debugging. All AI-assisted content, research decisions, implementations, analyses, and reported results were independently reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

In this work, we used generative AI tools to provide feedback on research methodology and experimental design, assist with software debugging and code editing, support the interpretation and presentation of experimental results, and assist with literature organization and manuscript drafting and editing. AI assistance also supported numerical presentation during drafting; such assistance does not establish experimental validity. The authors are responsible for checking reported results against execution records and for validating all AI-assisted decisions, code, analyses, and text. The authors take full responsibility for the final content of this work, including all claims, experimental results, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies learning algorithms for tool-using LLMs using benchmark environments. Our experiments do not involve human subjects, private user data, or real-world external actions. As with other autonomous tool-use systems, deploying such methods in real-world environments could introduce risks if a model is allowed to perform incorrect or unauthorized actions. Practical deployments should therefore employ appropriate access controls, permission boundaries, sandboxing, and human oversight for consequential operations. All experiments in this work are conducted within controlled benchmark environments.

REPRODUCIBILITY STATEMENT

The appendix documents the implementation and evaluation protocol. Appendix B.1 reports the base model, training hyperparameters, interaction budgets, and retrieval configuration; Appendix B.2 describes the controlled implementation of all baselines; and Appendix B.3 describes the BFCL-v3 evaluation protocol and split sizes. Additional ablations and skill analyses are reported in Appendix C, and Appendix D provides a trace-grounded case study illustrating the complete skill lifecycle. All compared methods use matched acting models, tool interfaces, and evaluation budgets whenever applicable.

REFERENCES

Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. SkillCraft: Can LLM Agents Learn to Use Tools Skillfully? *arXiv preprint arXiv:2603.00718*, 2026.

Gen Dong, Yanjie Gao, Liqun Li, Tianyin Xu, Yu Hua, and Fan Yang. Agent Skills Can Be Harmful: An Empirical Study of Skill-Induced Failures in LLM Agents. *arXiv preprint arXiv:2608.11888*, 2026.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From Raw Experience to Skill Consumption: A Systematic Study of Model-Generated Agent Skills. *arXiv preprint arXiv:2605.23899*, 2026.

Yanzhou Li, Yiran Zhang, Xiaoyu Zhang, Xiaoxia Liu, and Yang Liu. CODESKILL: Learning Self-Evolving Skills for Coding Agents. *arXiv preprint arXiv:2605.25430*, 2026a.

Yixuan Li, Mingshu Cai, Ziyang Xiao, Wanyuan Wang, Yanchen Deng, and Bo An. MIND-Skill: Quality-Guaranteed Skill Generation via Multi-Agent Induction and Deduction. *arXiv preprint arXiv:2605.08670*, 2026b.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems 37*, pp. 126544–126565. Curran Associates, Inc., 2024.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 48371–48392. PMLR, 2025.

Kang Peng, Zhiwei Zhang, Yichen Zhang, Zezhong Wang, Yiming Du, Geng Tu, Baojun Wang, Bin Liang, Ruifeng Xu, and Kam-Fai Wong. Write, Execute, Refine: From Skill Followers to Skill Optimizers via Reinforcement Learning from Execution Feedback. *arXiv preprint arXiv:2608.17587*, 2026.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward Is All Tool Learning Needs. In *Advances in Neural Information Processing Systems 38*, pp. 105523–105553. Curran Associates, Inc., 2025.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems 36*, pp. 68539–68551. Curran Associates, Inc., 2023.

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. *arXiv preprint arXiv:2402.03300*, 2024.

Junhao Shen, Zhanqiu Zhang, Yiwen Guo, and Hong Cheng. Progressive Agent Skill Generation via Reinforcement Learning. *arXiv preprint arXiv:2608.01678*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024.

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. *arXiv preprint arXiv:2604.04804*, 2026a.

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. Association for Computational Linguistics, 2026b.

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.

Min Yang, Jinghua Piao, Xu Xia, Xiaochong Lan, Jiaju Chen, Yongshun Gong, and Yong Li. SkillMaster: Toward Autonomous Skill Mastery in LLM Agents. *arXiv preprint arXiv:2605.08693*, 2026a.

Shidong Yang, Ziyu Ma, Tongwen Huang, Xucong Wang, Renda Li, Yiming Hu, Yong Wang, and Xiangxiang Chu. SkillForge: Evolving Verifiable Skills for Reinforcement Learning Agents. *arXiv preprint arXiv:2608.24747*, 2026b.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Yue Liao, Hongru Wang, Mengyue Yang, Heng Ji, Michael Littman, Jun Wang, Shuicheng Yan, Philip Torr, and Lei Bai. The Landscape of Agentic Reinforcement Learning for LLMs: A Survey. *arXiv preprint arXiv:2509.02547*, 2025.

Qi Zhang, Zhaopeng Feng, Xiaonan Shi, Xiaomeng Hu, Chu Liu, Pengjun Xie, Xiaobin Wang, Jieping Ye, Bryan Hooi, Haobo Wang, and Junbo Zhao. SkillComposer: Learning to Evolve Agent Skills for Specification and Generalization. *arXiv preprint arXiv:2606.06079*, 2026.

Yingli Zhou, Shu Wang, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A Comprehensive Survey on Agent Skills: Taxonomy, Techniques, and Applications. *arXiv preprint arXiv:2605.07358*, 2026.

A Comparison with Representative Tool-Skill Methods

To further contextualize the three limitations discussed in Section 1, we compare A^3-Skill with representative tool-skill methods along three design dimensions. We focus on methods that explicitly study reusable skills for tool-using agents or evaluate skill learning in interactive tool-use environments such as AppWorld, BFCL, and τ^2-Bench. Here, **L1** characterizes where skill-management capabilities reside, **L2** describes whether skill-related supervised fine-tuning is used, and **L3** describes how reusable skills are represented. The comparison is intended to characterize system designs rather than to imply that any individual design choice is intrinsically defective.

Table 7: Comparison with representative tool-skill methods along the three design dimensions discussed in the introduction. Entries for prior work directly describe their corresponding design choices, while checkmarks in the final row highlight the target design choices of A^3-Skill.

Paradigm	Method	L1: Management	L2: Skill supervision	L3: Representation
Training-free	SkillCraft (Chen et al., 2026)	Prompted composition and reuse	None	Executable skill cache
Training-free	MIND-Skill (Li et al., 2026b)	Separate induction and deduction agents	Prompt optimization; no model SFT	Natural-language skill documents
Training-free	SkillX (Wang et al., 2026a)	External library builder and downstream agent	None	Hierarchical textual skill knowledge
Training-based	SAGE (Wang et al., 2026b)	Learned generation and use; externally structured library pipeline	Expert-experience SFT followed by RL	Executable skills
Training-based	SkillComposer (Zhang et al., 2026)	Separate skill composer and task executor	Rejection-sampled SFT	Natural-language skills
Training-based	SkillForge (Yang et al., 2026b)	RL actor; external teacher for induction and refinement	RL; no skill-related SFT	Textual skills injected through the skill bank
Training-based	WER (Peng et al., 2026)	Trainable skill optimizer with a frozen executor	RL from execution feedback	Natural-language skills
Training-based	Skill-α (Shen et al., 2026)	Trainable skill editor with a fixed downstream worker	SFT warm-up followed by GRPO	Natural-language `SKILL.md` documents
Training-based	**A^3-Skill**	**✓ Unified acting policy**	**✓ No skill-operation supervision**	**✓ Executable programs**

Table 7 distinguishes the original methods from our controlled baseline adaptations. Training-based skill systems can still separate skill creation or refinement from task execution, while SAGE additionally uses supervised initialization. A^3-Skill jointly targets unified skill management, learning without supervised skill operations, and executable representations.

B Experimental Details

B.1 Implementation Details

We use Qwen3-4B-Instruct-2507 as the base model for all controlled experiments. Both stages of A^3-Skill are optimized with GRPO, and all policy parameters are updated exclusively using interaction-derived rewards. Unless otherwise specified, all compared methods use the same AppWorld task splits, atomic tool interface, maximum interaction budget, and evaluation protocol.

Table 8 summarizes the main training hyperparameters. Values shared between compared methods are held fixed whenever applicable.

Configuration	Value
Base model	Qwen3-4B-Instruct-2507
RL algorithm	GRPO
Stage-I training steps	160
Stage-II training steps	120
Rollouts per prompt	4
Learning rate	1×10^{-6}
Maximum turns per task	5
Sampling temperature	1.0
Maximum generation length	2048 tokens
Retrieved skills per task	Top-3
Training hardware	A6000

Table 8: Implementation details of A^3-Skill.

Methods	Policy Training	Skill Source	Skill Format	External Teacher	Test-Time Teacher
Direct	–	–	–	–	–
ToolRL	RL	–	–	–	–
SkillCraft	–	A^3-Skill library	Executable	–	–
MIND-Skill	–	Expert-generated	Executable	✓	–
SAGE	SFT + RL	Self-managed	Executable	✓	–
A^3-Skill	**RL only**	**Self-managed**	**Executable**	–	–

Table 9: Controlled implementation of the baselines. The SkillCraft, MIND-Skill, and SAGE rows adapt the corresponding training or deployment paradigms to our executable-skill environment rather than reproducing all components of the original systems.

B.2 Controlled Baseline Implementations

The baselines in the main experiments are controlled adaptations of prior training or deployment paradigms rather than verbatim reproductions of their original systems. This design keeps the acting model, tool environment, and evaluation protocol fixed while varying the source of skill knowledge and skill-management supervision. Table 9 summarizes the resulting comparison.

ToolRL. ToolRL represents task-only tool-use reinforcement learning. We optimize the same Qwen3-4B policy on the same training tasks using task-outcome rewards, but remove the skill library and all skill-management operations. This baseline isolates the benefit of reusable skill learning from the benefit of RL-based tool-use training itself.

SkillCraft. SkillCraft represents prompting-only executable skill reuse. The frozen Qwen3-4B policy receives the same global executable skill library constructed by A^3-Skill, together with the same retrieval results and skill interfaces, but receives no skill-management training. This comparison isolates whether the learned library and prompting protocol alone are sufficient.

MIND-Skill. MIND-Skill represents expert-generated skill knowledge. A stronger teacher model is provided with successful training trajectories and constructs executable skills offline. The generated skills undergo the same executable validation used by A^3-Skill before being placed in the global library. At test time, the acting model remains the frozen Qwen3-4B policy, and no external teacher is queried. Thus, this baseline measures the benefit of higher-quality externally generated skills while controlling test-time model capacity.

SAGE. SAGE represents expert-bootstrapped skill-management learning. We first construct supervised trajectories containing skill-management decisions and corresponding executable skills using an external teacher, and use them to initialize Qwen3-4B through SFT. The resulting model is

then optimized with GRPO under the same skill interface and environment as A^3-Skill. This provides a controlled comparison between expert-supervised SFT–RL training and our RL-only training.

B.3 Cross-Benchmark Protocol on BFCL-v3

We additionally evaluate cross-benchmark transfer using the Base Multi-Turn category of BFCL-v3. Following prior skill-learning evaluations, we divide the 200 instances into 50 training instances and 150 held-out test instances using a fixed split. The training subset is used for target-domain skill construction or model adaptation where required by each method, while all reported results are computed on the same held-out test set. Because the model may receive target-benchmark training instances before evaluation, we refer to this experiment as *cross-benchmark generalization* rather than strict zero-shot out-of-domain evaluation.

All methods are evaluated on the same held-out subset; any target-domain adaptation uses only the training subset.

C Additional Experimental Results

C.1 Additional Ablations

Reward design. We next study the reward components used to distinguish reusable skills from skills that are merely syntactically valid. Removing verified reuse utility causes the largest degradation, indicating that successful downstream use provides a more informative signal than construction validity alone. Workflow alignment further helps favor reusable procedural abstractions over task-specific programs.

Variants	Sequential TGC
A^3-Skill	**29.76**
w/o reuse utility	25.00
w/o workflow alignment	26.79
w/o delayed construction credit	25.60

Table 10: Ablation of the skill-learning reward design.

C.2 Additional Skill Analysis

We provide additional analyses of executable validation and retrieved-skill compatibility. All values are percentages unless otherwise specified.

Executable validation. We measure how often attempted skill construction or revision passes executable validation. Table 11 reports only aggregate validation rates because the number of UPDATE attempts can be small for individual variants and therefore yields unstable per-operation percentages. A^3-Skill achieves substantially higher validation rates than the frozen SkillCraft policy, indicating that RL improves the quality of generated skill programs rather than simply increasing the frequency of library modifications.

Methods	Sequential	Cold-Start
SkillCraft	7.89	18.18
A^3-Skill	**50.00**	**46.43**

Table 11: Executable validation pass rate (%).

Robustness to mismatched retrieved skills. To examine whether A^3-Skill learns to selectively reuse transferred skills, we divide retrieved candidates into equally sized high-, medium-, and low-compatibility groups according to a combined semantic and workflow compatibility score. The

policy itself uses semantic-only retrieval. Table 12 reports Skill Utilization, Successful Reuse, and the frequency of subsequent UPDATE or BUILD decisions within each group.

As candidate compatibility decreases, the policy becomes substantially less likely to execute the retrieved skill. Moreover, even when a low-compatibility skill is executed, it is less likely to contribute to successful task completion. Conversely, the frequency of UPDATE and BUILD decisions increases as compatibility decreases. These results suggest that the learned policy does not indiscriminately reuse retrieved skills, but instead shifts from direct reuse toward adaptation or reconstruction when transferred skills become less compatible.

Compatibility	**Skill Utilization**	**Successful Reuse**	**Update/Build Rate**
High	50.00	27.00	18.00
Medium	25.00	15.00	34.00
Low	9.00	5.00	45.00

Table 12: Skill-management behavior (%) under different levels of retrieved-skill compatibility. Compatibility groups are equal-sized tertiles based on the combined semantic and workflow compatibility score.

D CASE STUDY

We use a cold-start scenario to illustrate the complete lifecycle of an executable tool skill. All three tasks share the same underlying workflow: importing all Markdown files from a specified File System directory into Simple Note, while the source directory changes across tasks. The example is trace-grounded in the recorded interaction on tasks A, B, and C.

Task A: acquiring a reusable workflow. The scenario starts with an empty local skill library.

```
[TASK A]
directory_path =
   "~/documents/personal/notes/"

[LOCAL SKILL LIBRARY]
empty

[AGENT -> ATOMIC TOOLS]
show_directory(directory_path=...)
show_file(...)
create_note(...)
...
complete_task(status="success")

[ENVIRONMENT]
Task completed successfully.

[POST-TASK DECISION]
BUILD import_markdown_notes_to_simple_note
```

From the successful tool trajectory, the policy extracts the recurring File-System-to-Simple-Note workflow and parameterizes the source directory. The resulting skill has the following reusable interface:

```
skill_name:
   import_markdown_notes_to_simple_note

inputs:
   directory_path: str

outputs:
   imported_note_count: int

workflow:
   list Markdown files
   -> read each file
   -> derive title from filename
   -> create one Simple Note note per file
```

The initial implementation, however, contains a localized input-binding defect:

```
directory_path = inputs.directory_path

file_entries = apis.file_system.show_directory(
   access_token=file_system_access_token,
   directory_path=directory_path,
   entry_type="files",
)

for file_entry in file_entries:
   ...
```

Task B: adapting the skill from execution feedback. Task B requires the same workflow with a different directory.

```
[TASK B]
directory_path =
   "~/documents/personal/simple_note/"

[RETRIEVED / LOCAL SKILL]
import_markdown_notes_to_simple_note

[AGENT -> SKILL]
execute_skill(
   "import_markdown_notes_to_simple_note",
   inputs={
      "directory_path":
      "~/documents/personal/simple_note/"
   },
)

[SKILL EXECUTION]
AttributeError:
'dict' object has no attribute 'directory_path'
```

The failure does not invalidate the reusable workflow: the task still requires the same file enumeration, file reading, title transformation, and note creation procedure. Instead, the execution feedback identifies a concrete defect in the skill implementation. The appropriate lifecycle decision is therefore UPDATE.

```
[POST-TASK DECISION]
UPDATE import_markdown_notes_to_simple_note

# Minimal repair
- directory_path = inputs.directory_path
+ directory_path = inputs["directory_path"]
```

After the repair, the skill preserves its original contract and workflow while correcting only the failed input binding.

```
[UPDATED SKILL]
import_markdown_notes_to_simple_note

[VALIDATION]
contract: pass
execution: pass
```

Task C: reusing the adapted skill. The third task again changes only the source directory.

```
[TASK C]
directory_path =
   "~/documents/personal/personal_notes/"

[LOCAL SKILL]
import_markdown_notes_to_simple_note
   status = validated

[AGENT -> SKILL]
execute_skill(
   "import_markdown_notes_to_simple_note",
   inputs={
      "directory_path":
      "~/documents/personal/personal_notes/"
   },
)

[SKILL OUTPUT]
{
   "status": "success",
   "imported_note_count": ...
}

[AGENT -> ENVIRONMENT]
complete_task(status="success")

[ENVIRONMENT]
Task completed successfully.

[POST-TASK DECISION]
KEEP
```

This trajectory illustrates the intended skill lifecycle: task A provides a successful workflow from which a reusable executable skill is acquired; task B exposes a localized implementation defect and supplies the evidence required to revise it; and task C directly reuses the repaired skill without regenerating its internal tool-use procedure. The example also highlights why adaptation is important for executable skills: a skill can capture the correct high-level workflow while still requiring a small execution-grounded repair before reliable reuse.