SELF-IMPROVING SKILLS FROM FAILURE: LEARNING TO RECOVER, REMEMBER, AND EVOLVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent frameworks increasingly package procedural knowledge as reusable *skills* that are selected from a skill library and executed on demand. Yet even a correctly selected skill may fail in execution, and such failures are typically treated as terminal outcomes rather than reused for improvement. We propose a **skill-augmented agent system** that learns from failures through reflective recovery, reusable case memory, and persistent skill revision, enabling the model and skill library to co-evolve through interaction. Optimizing this system is challenging because outcome-level reinforcement learning cannot distinguish credit across skill selection, execution, and reflection. We therefore introduce **FIRE** (**F**ailure-**I**nformed **R**einforcement and **E**volution), which performs stage-aware credit assignment for these components. Across four in-distribution task families and three out-of-distribution benchmarks, FIRE achieves strong skill-routing and task performance across Qwen3-4B and Llama-3.1-8B. With Qwen3-4B, FIRE reaches 95.1% Skill Hit, 92.2% Task Success, and 88.0% Joint Success in-distribution, together with 35.4% average OOD Task Success.

1 INTRODUCTION

Large language models (LLMs) are capable of reasoning, using tools, and interacting with environments to solve complex tasks (Yao et al., 2023; Schick et al., 2023). Skill-augmented agent systems have attracted growing attention due to their ability to organize procedural knowledge into reusable skills that can be retrieved and loaded on demand (Li et al., 2026; Shi et al., 2026; Xia et al., 2026). As skill libraries grow, deciding which skill to use becomes an explicit decision of the agent rather than a fixed preprocessing step. This creates a coupled challenge: the agent must not only select an appropriate skill, but also execute it correctly. Even a correctly selected skill may fail if its procedure is misapplied, an intermediate decision is incorrect, or it is poorly adapted to the current task. Yet such failures are typically reduced to terminal outcomes, leaving the information they reveal largely unused.

Existing work has explored several ways to learn from skill failures, but these capabilities are typically developed as separate adaptation mechanisms. SkillGate (Li et al., 2026) improves credit assignment for skill selection; Reflexion (Shinn et al., 2023) and PivoARL (Guo et al., 2026) use failed attempts for reflection and retry; ExpeL (Zhao et al., 2024) and SkillRL (Xia et al., 2026) reuse historical experience; and SkillRevise (Liu et al., 2026) improves skills from execution evidence. Together, these studies show that failures can support recovery, experience reuse, and skill improvement, yet these functions remain largely disconnected: a failure that repairs the current attempt is rarely retained as reusable experience and further accumulated to improve future skills. This motivates a skill-augmented agent that can recover through reflection, store experience in memory, and recursively improve itself.

Optimizing such a skill-augmented agent system is not trivial since skill selection, execution, and reflection share the same outcome rewards, introducing the challenge of credit assignment, whereas reinforcement learning with outcome rewards provides coarse credit over the complete trajectory. A correct skill selection may still be followed by an incorrect execution, while a reflection decision may either retain the current skill and repair its execution or replace it with a better one. Assigning the same outcome-derived learning signal across these decisions therefore obscures which behavior should be reinforced and which should be corrected. This mismatch motivates an optimization

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.

method that assigns learning signals according to the role of each decision and applies them to the corresponding tokens.

To this end, we propose a failure-driven **skill-augmented agent system** and **FIRE**, a stage-aware reinforcement-learning method for optimizing it. The system integrates reflective recovery, a reusable case library, and persistent skill revision: reflection repairs failed executions, the case library stores evaluated trajectories as reusable routing and execution experience, and skill revision accumulates recurring failures to improve skill descriptions and contents. Together, these mechanisms allow failures to support recovery, memory, and continued skill evolution across interactions. To optimize this system, **F**ailure-**I**nformed **R**einforcement and **E**volution (**FIRE**) performs stage-aware token-level credit assignment for skill selection, execution, and reflection. Across four in-distribution task families and three out-of-distribution benchmarks, FIRE achieves strong skill-routing and task performance with both Qwen3-4B and Llama-3.1-8B. With Qwen3-4B, FIRE reaches 95.1% Skill Hit, 92.2% Task Success, and 88.0% Joint Success in-distribution, together with 35.4% average OOD Task Success. In summary, our contributions are as follows:

- We propose a **skill-augmented agent system** that learns from failures through reflective recovery, reusable case memory, and recursive skill evolution, enabling the model and skill library to co-evolve through interaction.
- We introduce **FIRE**, which optimizes the agent system with **stage-aware credit assignment**, providing separate learning signals for skill selection, execution, and reflection.
- Extensive experiments across multiple task families, OOD benchmarks, and model backbones demonstrate strong performance in skill routing and task execution, with controlled ablations validating the proposed agent components and credit-assignment design.

2 Related Work

In this section, we review two relevant topics, skill-augmented agents and learning agents from failure and experience.

Skill-augmented agents. LLM agents increasingly use reusable skills to store procedural knowledge and support complex decision making. Voyager (Wang et al., 2024) maintains an ever-growing library of executable skills acquired through interaction, while SAGE (Wang et al., 2026a) incorporates skill generation and utilization into reinforcement learning. More recent work studies different components of skill learning: SkillGate (Li et al., 2026) addresses credit assignment for in-policy skill selection; Skill1 (Shi et al., 2026) jointly optimizes skill selection, utilization, and distillation; and SkillRL (Xia et al., 2026) recursively co-evolves a skill library with the agent policy. Skill-R1 (Vishe et al., 2026) performs recurrent skill optimization from verified rollout outcomes, while SkillRevise (Liu et al., 2026) iteratively revises skills from execution evidence. Our work instead focuses on a failure-driven skill-augmented agent that integrates reflective recovery, reusable experience, and recursive skill improvement within a closed learning loop.

Learning agents from failure and experience. A related line of work improves language agents by exploiting feedback and experience from previous attempts. Self-Refine (Madaan et al., 2023) iteratively improves outputs through self-generated feedback, while Reflexion (Shinn et al., 2023) converts feedback into verbal reflections stored in episodic memory. ExpeL (Zhao et al., 2024) distills reusable insights from past task experience, and Agent Workflow Memory (Wang et al., 2025) induces reusable workflows from agent trajectories to guide subsequent interactions. PivoARL (Guo et al., 2026) uses structured reflection to identify pivotal errors and performs local retry from the corresponding states. Building on these directions, our agent uses failures for reflective recovery, experience reuse, and skill revision, while FIRE further provides stage-aware credit assignment for skill selection, execution, and reflection.

3 Preliminaries

Our method builds on two components: a skill-augmented agent setting, which defines how skills are retrieved, selected, and executed, and Group Relative Policy Optimization (GRPO), which provides the underlying reinforcement-learning objective.

Skill-Augmented LLM Agents. We consider a task $x \sim \mathcal{D}$ and a reusable skill library $\mathcal{S} = \{s_1, \ldots, s_M\}$. Following prior work (Li et al., 2026; Shi et al., 2026; Xia et al., 2026), each skill is represented as $s = (\mathrm{id}(s), \mathrm{desc}(s), \mathrm{body}(s))$. A retriever produces a task-specific candidate set $\mathcal{C}_x = \mathrm{Retrieve}(x, \mathcal{S})$ with $|\mathcal{C}_x| = K$. The policy π_θ selects $\hat{\mathcal{S}}_x \subseteq \mathcal{C}_x$ from skill identifiers and descriptions, after which the corresponding skill bodies are revealed for execution. Let $\mathcal{G}_x \subseteq \mathcal{C}_x$ denote the ground-truth skill set and $\mathcal{V}(x, y) \in \{0, 1\}$ the verifier for output y.We adopt this setting throughout the paper. It naturally separates the agent's decisions into which skill to select and how to execute the selected skill, which forms the basis of our reflective interaction and stage-aware credit assignment.

Group Relative Policy Optimization. We build on Group Relative Policy Optimization (GRPO) (Shao et al., 2024). For each task x, GRPO samples G trajectories $\{\tau_i\}_{i=1}^{G}$ with rewards R_i and computes $A_i = (R_i - \mu_R)/(\sigma_R + \epsilon)$, where μ_R and σ_R are the group reward mean and standard deviation. Let $\rho_{i,t} = \pi_\theta(y_{i,t} \mid y_{i,<t})/\pi_{\mathrm{old}}(y_{i,t} \mid y_{i,<t})$. GRPO optimizes

$$\mathcal{L}_{\mathrm{GRPO}} = -\mathbb{E}_{i,t}\left[\min\left(\rho_{i,t}A_i,\ \mathrm{clip}(\rho_{i,t}, 1-\epsilon_{\mathrm{lo}}, 1+\epsilon_{\mathrm{hi}})A_i\right)\right] + \beta\mathcal{L}_{\mathrm{KL}}. \tag{1}$$

We retain this clipped policy-optimization objective in FIRE. However, instead of broadcasting a single trajectory-level advantage to all tokens, we construct stage-aware advantages for the corresponding selection, execution, and reflection tokens, as introduced in Section 5.

4 METHODOLOGY

Our method consists of two coupled components, as illustrated in Figure 1. The **skill-augmented agent system** performs *Select–Execute–Reflect* interaction with an evolving Skill Library and a reusable Case Library, while **FIRE** optimizes the agent through stage-aware credit assignment and improves the Skill Library through failure-driven revision.

System and Overall Objective. At training iteration t, the system maintains model parameters θ_t, Skill Library $\mathcal{S}_t$, and Case Library $\mathcal{M}_t$. Given evaluated trajectories $\mathcal{B}_t$, it updates $(\theta_t, \mathcal{S}_t, \mathcal{M}_t) \xrightarrow{\mathcal{B}_t} (\theta_{t+1}, \mathcal{S}_{t+1}, \mathcal{M}_{t+1})$: FIRE updates θ through stage-aware credit assignment, failure evidence revises $\mathcal{S}$, and evaluated episodes are appended to $\mathcal{M}$. The updated model and libraries are then reused in subsequent interactions.

4.1 SKILL-AUGMENTED AGENT SYSTEM

We reuse the notation introduced in Section 3 and define new variables when they first appear.

Reflective Skill Interaction. For each task x, the agent retrieves relevant historical cases $\mathcal{R}_x = \mathrm{RetrieveCase}(x, \mathcal{M}_t)$ in addition to the candidate skill set $\mathcal{C}_x$, where $\mathcal{M}_t$ is the Case Library at iteration t and $\mathcal{R}_x$ denotes the retrieved cases. These cases provide reusable routing and execution experience. The interaction is

$$\begin{aligned} S_1 &\sim \pi_\theta(\cdot \mid x, \mathcal{C}_x, \mathcal{R}_x), \quad y_1 \sim \pi_\theta(\cdot \mid x, \mathrm{body}(S_1), \mathcal{R}_x), \quad v_1 = \mathcal{V}(x, y_1), \\ (\xi, a, S_2) &\sim \pi_\theta(\cdot \mid x, \mathcal{C}_x, S_1, y_1, \mathcal{R}_x), \quad \text{if } v_1 = 0, \\ y_2 &\sim \pi_\theta(\cdot \mid x, \mathrm{body}(S_2), \mathcal{R}_x, y_1, \xi), \quad v_2 = \mathcal{V}(x, y_2). \end{aligned} \tag{2}$$

Here, S_1 is the initial skill configuration, y_1 its execution output, and $v_1 \in \{0, 1\}$ the verifier outcome. If $v_1 = 1$, the episode terminates; otherwise, ξ is a structured reflection containing *failure attribution*, *diagnosis*, and a *recovery strategy*, and $a \in \{\textsc{Keep}, \textsc{Replace}\}$ is the recovery decision. For KEEP, $S_2 = S_1$; for REPLACE, S_2 is the revised skill configuration. The same *Execute* component is then invoked again to produce y_2, with v_2 denoting its recovery outcome. Conditioning this execution on both y_1 and ξ makes recovery failure-aware rather than an independent retry.

Reusable Case Memory. After evaluation, the complete interaction trajectory τ_x is compressed into a reusable case $m_x = \mathrm{Case}(\tau_x)$ and appended as $\mathcal{M}_{t+1} = \mathcal{M}_t \cup \{m_x\}$, where m_x denotes its case representation. Each case records the task, selected skills, execution outcomes, and, when available, failure diagnosis and recovery behavior. The append-only Case Library therefore exposes accumulated experience to subsequent tasks.

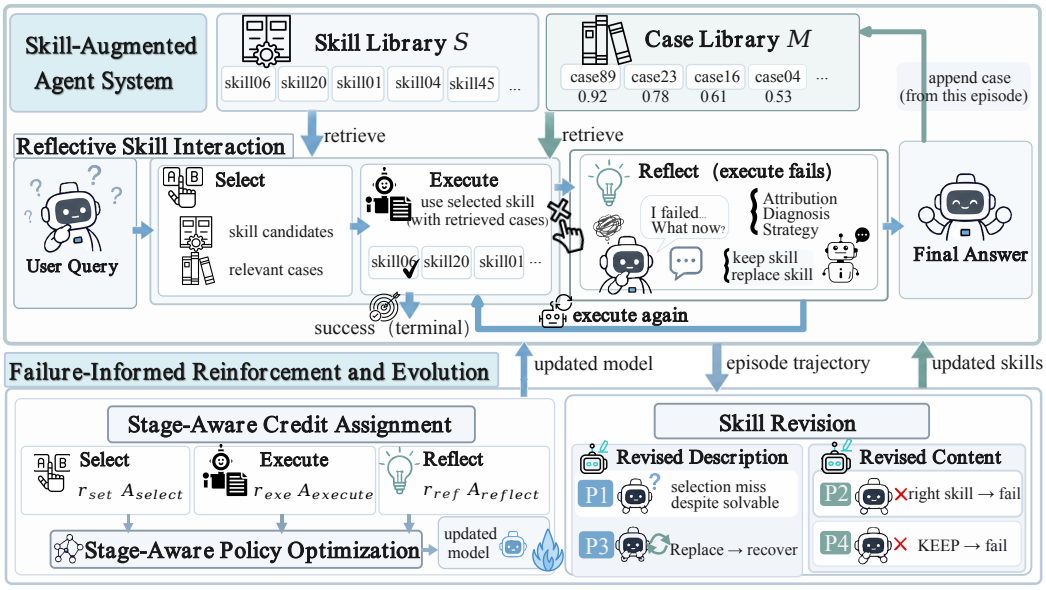

Figure 1: Overview of our method. **Top:** the skill-augmented agent performs *Select–Execute–Reflect* interaction with reusable skill and case libraries, enabling failure recovery and experience reuse. **Bottom:** FIRE provides stage-aware credit assignment and failure-driven skill revision to jointly improve the model and skill library.

Evolvable Skill Library. Each skill contains a description used for routing and executable content used for task solving, as defined in Section 3. We revise these fields with different evidence: routing failures update skill descriptions, whereas execution failures update skill contents. The resulting Skill Library is reused in subsequent episodes so that past failures can improve future selection and execution.

4.2 FIRE: Failure-Informed Reinforcement and Evolution

FIRE learns from generated trajectories at two levels: it assigns component-specific credit to *Select*, *Execute*, and *Reflect* at the model level, and converts recurring routing and execution failures into persistent Skill Library revisions at the skill level.

Stage-Aware Credit Assignment. The three components have distinct responsibilities: *Select* chooses the skill, *Execute* applies it, and *Reflect* determines how a failed execution should be repaired. A single trajectory-level reward therefore obscures which component should receive credit.

Let $\mathcal{G}_x$ denote the ground-truth skill set defined in Section 3. For a selected skill set S, we measure routing quality by the set-level F1 score $\Phi(S, \mathcal{G}_x)$ and define the component rewards as

$$\Phi(S, \mathcal{G}_x) = \frac{2|S \cap \mathcal{G}_x|}{|S| + |\mathcal{G}_x|}, \qquad r_{\text{sel}} = \Phi(S_1, \mathcal{G}_x),$$
$$r_{\text{exe}}^{(j)} = \mathcal{V}(x, y_j), \quad j \in \{1, 2\}, \qquad r_{\text{ref}} = \begin{cases} v_2, & a = \textsc{Keep}, \\ \Phi(S_2, \mathcal{G}_x) - \Phi(S_1, \mathcal{G}_x), & a = \textsc{Replace}. \end{cases} \tag{3}$$

We set $\Phi = 0$ when its denominator is zero. Here, r_{sel} measures routing quality, $r_{\text{exe}}^{(j)}$ evaluates the j-th invocation of *Execute*, and r_{ref} evaluates the recovery decision. Both executions belong to the same semantic *Execute* component and share its reward channel. For reflection, KEEP is rewarded by successful recovery because it attributes the failure to execution, whereas REPLACE is rewarded by improved routing quality because it attributes the failure to skill selection.

Stage-aware Policy Optimization. Let $c \in \{\text{sel}, \text{exe}, \text{ref}\}$ index the three semantic credit channels and i index rollouts within a GRPO group. For rollout i, $r_i^{(c)}$ denotes the reward of channel c,

while μ_c and σ_c denote its group-wise mean and standard deviation. Let $\mathcal{T}_i^{(c)}$ be the generated-token positions assigned to channel c. FIRE computes

$$A_i^{(c)} = \frac{r_i^{(c)} - \mu_c}{\sigma_c + \epsilon}, \qquad n_c = \sum_i |\mathcal{T}_i^{(c)}|, \quad N = \sum_c n_c,$$
$$w_c = \frac{N}{n_c}, \qquad \widetilde{A}_{i,t} = w_c A_i^{(c)}, \qquad t \in \mathcal{T}_i^{(c)}. \tag{4}$$

where ϵ is a small constant for numerical stability, n_c is the number of generated tokens assigned to channel c, N is the total number of generated tokens across channels, and w_c balances their aggregate optimization strength. Channel-wise normalization prevents one component from determining another's credit, while token-length balancing prevents shorter components such as *Select* and *Reflect* from being overwhelmed by longer execution traces. We substitute $\widetilde{A}_{i,t}$ for the trajectory-level advantage in Equation 1, leaving the underlying clipped GRPO objective unchanged.

Failure-Driven Skill Revision. Model optimization improves how skills are used, while recurring failures can also expose deficiencies in the skills themselves. For each skill s, FIRE therefore maintains routing evidence $\mathcal{E}_{\text{route}}(s)$ and execution evidence $\mathcal{E}_{\text{exe}}(s)$. Routing evidence is collected when an appropriate available skill is missed or when REPLACE changes the skill configuration and enables successful recovery; execution evidence is collected when an appropriate skill is selected but execution fails, including unsuccessful recovery after KEEP.

The two forms of evidence update the corresponding skill fields:

$$\Delta d_s = \text{ReviseDesc}(s, \mathcal{E}_{\text{route}}(s)), \ \Delta b_s = \text{ReviseBody}(s, \mathcal{E}_{\text{exe}}(s)), \ s_{t+1} = s_t \oplus (\Delta d_s, \Delta b_s). \tag{5}$$

Here, Δd_s is the proposed update to the description of skill s, Δb_s is the proposed update to its executable content, and $\oplus$ applies the available revisions while leaving unchanged fields intact. Thus, routing failures improve future skill selection, execution failures improve future skill use, and the revised Skill Library is reused in subsequent episodes, closing the model–memory–skill feedback loop.

5 EXPERIMENTS

We report the experimental setup, main results, ablations, and analyses of transfer, training dynamics, and recursive recovery.

5.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate on seven task settings, including four in-distribution (ID) tasks from SRA-Bench (Su et al., 2026): THEOREMQA (Chen et al., 2023), LOGICBENCH (Parmar et al., 2024), MEDCALC-BENCH (Khandekar et al., 2024), and CHAMP (Mao et al., 2024), as well as three out-of-distribution (OOD) benchmarks: BIGCODEBENCH (Zhuo et al., 2024), R3-SKILL (Wang et al., 2026b), and SKILLEVAL (Tan et al., 2026). The ID tasks cover theorem application, logical reasoning, medical calculation, and mathematical reasoning. We use an 8:1:1 train/validation/test split for the ID data and select the best checkpoint on validation. We report ID results on the held-out test set and evaluate the same checkpoint on three OOD benchmarks without further adaptation.

Baselines. We evaluate all methods with two LLM backbones: QWEN3-4B-INSTRUCT-2507 (Yang et al., 2025) and LLAMA-3.1-8B-INSTRUCT (Grattafiori et al., 2024). We first include controlled baselines: **Base Model** without RL training, **Skill-Free RL** without skill conditioning, **Oracle-Skill RL** with ground-truth skills provided to the executor, and **Full-Skill RL** with all 16 candidate skills exposed during execution. We further compare against **GRPO** (Shao et al., 2024), applied to the complete selection–execution trajectory; **Skill-R1** (Vishe et al., 2026); **SkillRL** (Xia et al., 2026), **SkillRL (adapted)** (Xia et al., 2026), with skill revision disabled under our interaction protocol; and **SkillGate** (Li et al., 2026), reproduced under the same candidate setting. **SkillGate + Retry** additionally receives one retry after failure to control for FIRE's extra recovery opportunity. Finally, **FIRE w/o Mem. & Rev.** removes the case library and persistent skill revision, while **FIRE** denotes the complete framework.

Table 1: Main ID and OOD results (%). **Bold** and underlined denote the best and second-best non-oracle aggregate results for each backbone.

	In-Distribution (ID)							Out-of-Distribution (OOD)				
Method	**TQA**	**Logic**	**Med**	**CHAMP**	**Skill Hit**	**Task Success**	**Joint Success**	**BCB Task**	**R3 Skill**	**SE Task**	**Skill Hit**	**Task Success**
Qwen3-4B-Instruct												
Base Model	78.4	75.0	75.5	73.9	69.6	76.0	55.1	51.7	65.9	4.0	84.0	4.0
Skill-Free RL	68.9	75.0	25.5	82.6	–	54.8	–	23.0	66.2	6.0	–	14.5
Oracle-Skill†	62.2	96.1	87.3	87.0	100.0	83.0	83.0	51.2	67.5	15.0	89.2	33.1
Full-Skill RL	71.6	81.6	54.5	78.3	–	68.2	–	40.4	66.7	7.0	–	23.7
GRPO	67.6	89.5	82.7	73.9	68.2	79.9	56.2	42.9	74.3	11.0	73.0	27.0
Skill-R1	77.0	88.2	83.6	82.6	68.9	83.0	59.7	46.3	56.5	13.0	69.9	29.7
SkillRL*	86.5	92.1	90.9	73.9	74.9	88.7	67.8	53.3	67.4	6.0	75.7	29.6
SkillGate	68.9	92.1	85.5	78.3	86.6	82.3	74.6	45.1	77.6	3.0	73.3	24.1
SkillGate + Retry	81.1	98.7	84.5	69.6	70.7	86.2	61.5	51.8	72.1	12.0	83.1	31.9
FIRE w/o M&R	83.8	100.0	90.9	82.6	86.9	90.8	79.5	54.9	70.9	9.0	**85.5**	32.0
FIRE	85.1	100.0	93.6	82.6	**95.1**	**92.2**	**88.0**	55.8	76.8	15.0	83.4	**35.4**
Llama-3.1-8B-Instruct												
Base Model	36.91	46.05	57.73	0.00	68.20	44.52	36.40	16.2	43.4	0.0	42.4	8.1
Skill-Free RL	36.00	92.11	45.45	31.82	–	54.42	–	13.6	35.9	1.0	–	7.3
Full-Skill RL	24.83	89.47	18.64	40.00	–	40.99	–	3.1	22.7	2.0	–	2.5
GRPO	48.00	86.84	73.64	27.27	75.97	66.78	52.65	25.4	75.9	1.0	77.8	13.2
Skill-R1	43.62	82.89	70.91	37.78	66.96	64.31	47.70	12.3	37.6	3.0	36.8	7.6
SkillRL*	55.70	93.42	65.00	46.67	74.56	68.73	50.00	23.7	72.1	3.0	82.1	13.3
SkillGate	54.36	84.87	70.91	46.67	87.63	68.37	62.37	24.1	78.6	1.0	78.1	12.6
SkillGate + Retry	51.68	99.34	66.82	51.11	79.86	70.32	54.06	18.4	68.9	1.0	75.3	9.7
FIRE w/o M&R	57.05	98.03	78.64	46.67	87.81	**75.62**	**67.67**	29.4	73.7	4.0	82.2	16.7
FIRE	55.03	94.74	68.64	42.22	**93.11**	69.96	66.43	32.9	72.2	6.0	**86.0**	**19.4**

TQA, Logic, Med, and CHAMP denote TheoremQA, LogicBench, MedCalc-Bench, and CHAMP. BCB, R3, and SE denote BigCodeBench, R3-Skill, and SkillEval, respectively.
Skill Hit and Joint Success are undefined for Skill-Free RL and Full-Skill RL, which do not explicitly select a skill subset.

Evaluation metrics. We use three evaluation metrics. **Skill Hit** measures the fraction of instances whose initial skill selection contains at least one ground-truth skill; **Task Success** measures final task accuracy; and **Joint Success** requires both a successful initial skill hit and a correct final answer. For OOD evaluation, Skill Hit is macro-averaged over BigCodeBench, R3-Skill, and SkillEval, while Task Success is macro-averaged over BigCodeBench and SkillEval, since R3-Skill evaluates skill selection only. All task outcomes are evaluated with deterministic verifiers rather than LLM judges.

5.2 Main Results

We evaluate FIRE on both in-distribution (ID) and out-of-distribution (OOD) tasks and make three observations. **First, FIRE achieves strong in-distribution performance.** With Qwen3-4B, FIRE obtains the best non-oracle aggregate results, reaching 95.1% Skill Hit, 92.2% Task Success, and 88.0% Joint Success. On Llama-3.1-8B, FIRE again achieves the highest Skill Hit (93.11%), while FIRE w/o M&R attains higher Task Success, suggesting that the full system benefits routing more consistently than ID execution. **Second, FIRE demonstrates strong out-of-distribution generalization.** It achieves the highest OOD Task Success with both Qwen3-4B (35.4%) and Llama-3.1-8B (19.4%). OOD routing is more mixed: FIRE is slightly below its ablation on Qwen3-4B, but achieves the highest aggregate Skill Hit on Llama-3.1-8B. **Third, the gains transfer across model backbones.** Despite the performance gap between Qwen3-4B and Llama-3.1-8B, FIRE improves skill routing and yields the strongest OOD execution results in both settings.

5.3 Ablation Study

We conduct controlled ablations to examine three key design questions: how credit should be assigned across interaction stages, what makes reflective recovery effective, and how case memory interacts with persistent skill revision. Panels (a)–(c) of Table 2 isolate the optimization and recovery designs without the full skill-augmented agent system, while Panel (d) evaluates memory and skill revision within the proposed system.

Does FIRE need distinct credit for each interaction stage? Panel (a) studies credit assignment across selection, execution, and reflection. Individual channels exhibit clear specialization: selection credit primarily improves Skill Hit, whereas execution credit favors Task Success. Combining these signals yields more balanced performance. Moreover, separating the initial and recovery executions into independent advantages improves Task Success from 87.3% to 90.8% and Joint Success from 74.2% to 79.5% compared with sharing a single execution advantage, supporting finer-grained stage-aware credit assignment.

What makes reflective recovery effective? Panels (b)–(c) examine both the information used by reflection and the credit assigned to recovery decisions. Structured reflection achieves the strongest overall recovery, reaching 90.8% Task Success and 79.5% Joint Success. Removing either failure or reflection context degrades performance, while Retry Only achieves only 61.5% Joint Success, showing that an additional attempt alone is insufficient. Free-form reflection attains higher Skill Hit but substantially lower Task Success than structured reflection (84.1% vs. 90.8%), highlighting the importance of structured failure diagnosis. Panel (c) further shows that action-aligned recovery credit is beneficial: rewarding KEEP by recovery success and REPLACE by the improvement in routing quality ($\Delta\Phi$) achieves the strongest joint performance, whereas KEEP-only and mixed REPLACE rewards reduce Joint Success.

Do memory and skill revision work better together? Panel (d) studies the interaction between case memory and persistent skill revision under the same three-channel credit setting. Skill revision alone provides little overall benefit over removing both components, with Joint Success remaining nearly unchanged (73.9% vs. 74.2%) and CHAMP performance dropping substantially. Adding case memory together with skill revision increases Skill Hit from 84.1% to 95.1%, Task Success from 86.2% to 89.0%, and Joint Success from 73.9% to 85.9%. This indicates that persistent skill revision is substantially more effective when grounded in reusable experience accumulated across episodes.

5.4 Further Analyses

We further analyze FIRE through cross-model transfer, training dynamics, and recursive test-time recovery.

Cross-Model Transfer. We freeze the case and revised skill libraries learned with Qwen3-4B and attach them, without additional training, to seven models from Qwen2.5 (Yang et al., 2024), Qwen3, and Llama 3. All models are evaluated on the same 283 held-out tasks with deterministic verification. As shown in Table 3, FIRE libraries improve Skill Hit for all seven models and Joint Success for all except Qwen2.5-0.5B, with routing gains of $+39.58$ points on Qwen3-0.6B and $+20.14$ points on Llama-3.1-8B. Task Success also improves, but by smaller margins, suggesting that library transfer primarily strengthens routing while execution remains constrained by base-model capability.

Training Dynamics. Figure 2 tracks FIRE's recovery policy, performance, case memory, and skill revision during training. The policy rapidly shifts from exploring REPLACE to a stable KEEP-dominant regime, exceeding 90% throughout most later checkpoints. By step 200, 91.1% of failed initial executions had already selected a ground-truth skill, indicating that later failures increasingly stem from execution rather than routing. Meanwhile, Skill Hit and Joint Success improve over the variant without memory and revision, the case library grows continuously, and skill revisions rise rapidly before gradually saturating.

Recursive Test-Time Recovery. We freeze a checkpoint trained with one recovery round ($R = 1$) and evaluate $R \in \{0, 1, 2, 3\}$ on 566 held-out tasks without additional training. As shown in Table 4, Task Success rises monotonically from 83.92% at $R = 0$ to 89.58% at $R = 1$ and 90.64% at $R = 3$. Among 91 initial failures, FIRE cumulatively recovers 32, 33, and 38 cases for $R = 1, 2, 3$, increasing the recovery rate from 35.16% to 41.76%. Initial Skill Hit remains fixed at 93.99%,

Table 2: Ablation studies of FIRE (%). Panels (a)–(d) study credit assignment, reflection formulation, recovery reward, and the interaction between case memory and skill revision, respectively. In Panel (a), identical advantage labels indicate a shared credit channel, whereas distinct labels denote independently assigned credit.

					TQA	Logic	Med	CHAMP	Skill Hit	Task Success	Joint Success
(a) Credit assignment											
Variant	**Sel.**	**Exec$_1$**	**Ref.**	**Exec$_2$**							
Select only	A_1				60.8	90.8	84.5	52.2	81.3	77.4	65.7
Task only		A_1		A_1	70.3	90.8	89.1	78.3	64.3	83.7	56.5
Reflect only			A_1		64.9	86.8	86.4	52.2	66.4	78.1	53.0
Select + Task	A_1	A_2		A_2	67.6	92.1	89.1	65.2	**90.8**	82.3	76.3
Select + Reflect	A_1		A_2		63.5	93.4	89.1	56.5	88.3	80.9	72.8
All (3-ch.)	A_1	A_2	A_3	A_2	83.8	96.1	84.5	82.6	85.2	87.3	74.2
All (4-ch.)	A_1	A_2	A_3	A_4	83.8	100.0	90.9	82.6	86.9	**90.8**	**79.5**
(b) Reflection formulation											
No failure / reflection ctx.					71.6	94.7	89.1	60.9	87.3	83.7	73.5
No failure ctx.					77.0	96.1	83.6	87.0	89.4	85.5	77.7
No reflection ctx.					86.5	89.5	86.4	82.6	86.9	86.9	77.0
Free-form reflection					67.6	93.4	90.0	78.3	**91.5**	84.1	77.4
Retry only					81.1	98.7	84.5	69.6	70.7	86.2	61.5
Structured reflection					83.8	100.0	90.9	82.6	86.9	**90.8**	**79.5**
(c) Recovery reward											
KEEP only					87.8	96.1	89.1	87.0	84.1	90.5	77.4
Decoupled (ours)					83.8	100.0	90.9	82.6	**86.9**	**90.8**	**79.5**
Mixed REPLACE					87.3	96.1	88.2	87.0	84.1	90.3	77.0
(d) Case memory and skill revision											
Variant	**Case Memory**		**Skill Revision**								
w/o Mem., w/o Rev.					83.8	96.1	84.5	82.6	85.2	87.3	74.2
w/o Mem., w Rev.			✓		79.7	100.0	86.4	60.9	84.1	86.2	73.9
w Mem., w Rev.	✓		✓		86.5	100.0	84.5	82.6	**95.1**	**89.0**	**85.9**

Table 3: Cross-model transfer of frozen FIRE libraries (%). **Base** denotes the frozen model; **+ FIRE Libs.** adds the frozen case and revised skill libraries without additional training; Δ denotes absolute improvement in percentage points.

	Base			**+ FIRE Libs.**			**Δ (pp)**		
Model	**Skill Hit**	**Task Success**	**Joint Success**	**Skill Hit**	**Task Success**	**Joint Success**	**Skill Hit**	**Task Success**	**Joint Success**
Qwen2.5-0.5B-Instruct	0.35	0.00	0.00	0.71	1.06	0.00	+0.35	+1.06	+0.00
Qwen2.5-1.5B-Instruct	22.26	10.95	5.30	61.48	17.31	14.13	+39.22	+6.36	+8.83
Qwen2.5-3B-Instruct	14.49	1.06	1.06	26.15	2.83	2.12	+11.66	+1.77	+1.06
Qwen3-0.6B	31.45	33.22	14.13	71.02	38.52	31.80	+39.58	+5.30	+17.67
Qwen3-1.7B	59.72	58.66	45.23	83.75	64.66	57.24	+24.03	+6.01	+12.01
Llama-3.1-8B-Instruct	59.72	34.28	25.09	79.86	41.34	39.22	+20.14	+7.07	+14.13
Qwen3-4B-Instruct-2507	69.96	75.62	55.12	90.46	76.78	67.07	+20.50	+1.16	+11.95

confirming that the additional gains arise after initial selection and that the learned recovery behavior can be recursively reused beyond its training-time interaction depth.

5.5 CASE STUDY

We examine two representative MedCalcBench trajectories illustrating FIRE's two recovery modes; complete traces are provided in Appendix C. In Case A, FIRE attributes the failure to *selection*: the agent initially uses an unrelated APACHE II skill and an incorrect QT-correction formula. Reflection replaces it with the QTc Rautaharju skill, changing both the reasoning and answer from 202 to the correct 306; the recovered failure also contributes a persistent selection hint to the skill description. In Case B, the correct SOFA skill is already selected, but execution assigns 256.25 to the wrong

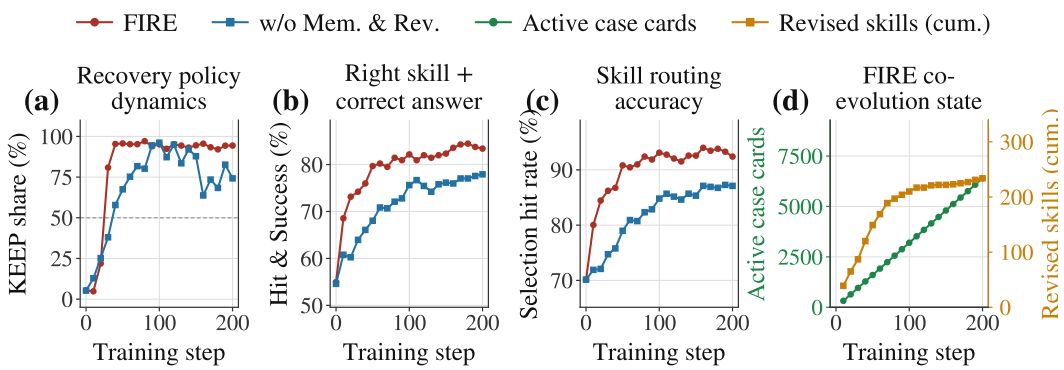

Figure 2: Training dynamics of FIRE. **(a)** Recovery decisions gradually shift toward a stable KEEP-dominant policy. **(b–c)** Joint Success and Skill Hit during training, compared with the variant without memory and skill revision. **(d)** Growth of the case library and the number of revised skills.

Table 4: Recursive test-time recovery of FIRE (%). The policy is trained with one recovery round (R=1) and evaluated with up to $R \in \{0, 1, 2, 3\}$ rounds without additional training. **Recovery Rate** denotes the fraction of the 91 initial failures that are eventually recovered.

Recovery Rounds R	**Model Calls**	**Task Success**	**Δ (pp)**	**Recovery Rate**	**Solved at Exec. 1/2/3/4**
0	2	83.92	—	0.00	475 / 0 / 0 / 0
1	4	89.58	+5.7	35.16	475 / 32 / 0 / 0
2	6	89.75	+5.8	36.26	475 / 32 / 1 / 0
3	8	90.64	+6.7	41.76	475 / 32 / 1 / 5

Table 5: Two representative reflective recovery trajectories.

Case	**Failure**	**Skill: Before $\rightarrow$ After**	**Action**	**Answer**
A	Selection	APACHE II $\rightarrow$ QTc Rautaharju	REPLACE	202 $\rightarrow$ 306
B	Execution	SOFA $\rightarrow$ SOFA	KEEP	8 $\rightarrow$ 9

scoring interval. FIRE instead attributes the error to *execution*, keeps the skill, and corrects the score from 8 to 9. Together, the cases show that FIRE does not mechanically replace skills after failure: reflection determines whether recovery requires rerouting or repairing execution.

6 CONCLUSION

We investigate how skill-augmented agents can systematically learn from failed skill-guided executions. We introduce a skill-augmented agent system that combines reflective recovery, reusable case memory, and persistent skill revision, allowing failures to improve both the current interaction and future skill use. FIRE optimizes this system through stage-aware credit assignment for skill selection, execution, and reflection. Experiments and controlled ablations across ID and OOD benchmarks with Qwen3-4B and Llama-3.1-8B validate the proposed recovery, memory, revision, and credit-assignment mechanisms. Further analyses demonstrate cross-model library transfer and recursive test-time recovery, highlighting the potential of failures as reusable supervision for jointly improving the model, memory, and skills.

Limitations and Future Work. Our current training formulation uses skill-level supervision to provide reliable routing credit. An important direction is to reduce this dependence by learning component-level credit directly from a shared task-level outcome signal, extending FIRE to settings where explicit skill annotations are unavailable.

AI USAGE STATEMENT

We used generative AI tools during the research process for brainstorming, discussing methodological and experimental choices, searching for relevant literature, and improving the clarity of the manuscript. Any references found with AI assistance were checked against the original sources before use.

AI tools were not used to determine experimental results or replace the evaluation procedures described in the paper. All research decisions, analyses, experiments, and final writing were reviewed and finalized by the authors.

REFERENCES

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.489.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Weiyang Guo, Zesheng Shi, Longhui Zhang, Zeen Zhu, Min Zhang, and Jing Li. Agent reinforcement learning via pivotal-aware self-feedback retry, 2026.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S. Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W. Safranek, Abid A. Anwar, Andrew Zhang, Aidan Gilson, Maxwell B. Singer, Amisha Dave, Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. MedCalc-Bench: Evaluating large language models for medical calculations. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Qingyao Li, Wenxiang Jiao, Shuai Shao, Kangning Zhang, Yuan Lu, Yi Guo, Weiwen Liu, Weinan Zhang, and Yong Yu. Skillgate: Training in-policy skill selection in long-horizon agents. *arXiv preprint arXiv:2608.18852*, 2026.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, Xiyu Ren, Chenyu Ruan, Haoran Li, and Yangqiu Song. Skillrevise: Improving llm-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Yujun Mao, Yoon Kim, and Yilun Zhou. CHAMP: A competition-level dataset for fine-grained analyses of LLMs' mathematical reasoning capabilities. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13256–13274. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.785.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.739.

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*, volume 36, pp. 68539–68551, 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. URL https://arxiv.org/abs/2402.03300.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 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, 2023.

Weihang Su, Jianming Long, Qingyao Ai, Yichen Tang, Changyue Wang, Yiteng Tu, and Yiqun Liu. Skill retrieval augmentation for agentic ai, 2026.

Zelin Tan, Yiqun Zhang, Hao Li, Zhiyao Cui, Hejia Geng, Shao Zhang, Hangfan Zhang, Yang Chen, Xiaosong Wang, Lilong Wang, Zhenfei Yin, Shuyue Hu, Chen Zhang, and Lei Bai. SKT: Skill-use training at scale via verified synthetic data generation, 2026.

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, 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.

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, 2026a. doi: 10.18653/v1/2026.acl-long.69.

Zifei Wang, Wei Wen, Qiang Ji, Ruizhi Qiao, and Xing Sun. Skill is not document: A query-conditional benchmark and two-stage retriever for llm agent skill routing, 2026b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 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, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

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. doi: 10.1609/aaai.v38i17.29936.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions, 2024.

A Additional Implementation Details

This section provides implementation details omitted from the main paper, including data statistics, optimization hyperparameters, credit assignment, case-memory construction, skill revision, and deterministic verification.

A.1 Task and Data Statistics

The global skill corpus contains $26{,}262$ reusable skills. For each task x, the retriever constructs a fixed candidate catalog $\mathcal{C}_x$ containing $K = 16$ skills. During selection, the policy observes only the skill identifier, name, and description; the corresponding skill body is revealed only after selection.

The training set contains $2{,}264$ tasks: 598 from TheoremQA, 608 from LogicBench, 880 from MedCalcBench, and 178 from CHAMP. We additionally use 283 validation and 283 test instances. The test split contains 74 TheoremQA, 76 LogicBench, 110 MedCalcBench, and 23 CHAMP instances. Most tasks require a single skill: among the training instances, $2{,}171$ have $|\mathcal{G}_x| = 1$, while 93 require multiple skills. For multi-skill tasks, the required cardinality $|\mathcal{G}_x|$ is provided to the selection policy; the same information is exposed to all compared methods.

A.2 Training Configuration

Table 6 summarizes the training configuration used for the main Qwen3-4B experiments.

Table 6: Training configuration used for FIRE.

Configuration	**Value**
Backbone	Qwen3-4B-Instruct-2507
Learning rate	1×10^{-6}
Rollouts per task	$G = 8$
Tasks per step	32 (256 trajectories)
PPO mini-batch size	128
Micro-batch per GPU	1
Maximum prompt / response length	8,192 / 10,240
Maximum rollout model length	20,480
Rollout temperature	1.0
KL coefficient	0.01
Gradient checkpointing	Enabled
Tensor parallelism	1
Validation frequency	Every 10 steps
Checkpoint frequency	Every 50 steps
Random seed	42
Hardware	1 node $\times$ 4 GPUs
Runtime	Ray + FSDP + vLLM
Maximum interaction steps	4

The main FIRE results with Qwen3-4B reported in the paper correspond to the checkpoint selected at training step 210.

A.3 CASE LIBRARY

During training, FIRE aggregates the $G = 8$ rollouts generated for the same task into a single case card rather than storing each rollout independently. A case records the number of successful and failed trajectories, initial gold-skill hits, recovery outcomes, frequently selected non-gold skills, representative successful strategies when available, and representative failure diagnoses.

New case cards are first written to a pending buffer and are published to the active Case Library every 10 training dumps. The active library remains frozen between publications, preventing a rollout group from retrieving experience generated by itself. Validation is read-only and never modifies either cases or skills.

Case retrieval is lexical and does not invoke an additional LLM. For a query task q and a stored case q', the retrieval score is

$$s(q, q') = 0.78\,\mathrm{sim}(q, q') + 0.12\,\mathbf{1}[\text{same data source}] + 0.10\,\mathrm{clip}(r_{\mathrm{gold}}, 0, 1),$$

where r_{gold} denotes the gold-selected success rate of the stored case. Cases with similarity below 0.08 are discarded. We use at most four cases for routing memory and two cases for execution memory.

Routing memory is available before skill selection and may contain both successful and failed historical cases. Execution memory is exposed only after skill selection and only contains cases with at least one successful gold-skill execution. All explicit `<answer>...</answer>` spans are removed before historical experience is inserted into a new prompt.

A.4 FAILURE-DRIVEN SKILL REVISION

FIRE maps recurring failures to four revision patterns, summarized in Table 7.

Table 7: Failure patterns used for persistent skill revision.

Pattern	Trigger	Revised field
P1: Selection bottleneck	Gold skill succeeds when selected, but is often missed	Description
P2: Content insufficient	Gold skill is selected but never succeeds	Content
P3: Replace recovery	REPLACE changes a non-gold selection to gold and succeeds	Description
P4: Execution deep error	KEEP is followed by another failed execution	Content

Revision proposals follow the priority P4 $>$ P3 $>$ P2 $>$ P1, and at most one proposal is generated for each skill in a revision cycle. Revisions are implemented as overlays:

$$s_{\mathrm{eff}} = s \oplus \Delta s,$$

so the original skill description and content are preserved. Description revisions append a concise routing hint, whereas content revisions append recurring pitfalls and recommended execution strategies.

A.5 DETERMINISTIC VERIFICATION

All task outcomes are evaluated using deterministic dataset-specific verifiers rather than LLM judges. The verifier first extracts the content inside `<answer>...</answer>`; if the span is absent, the full response is passed to the dataset evaluator as a fallback. TheoremQA uses type-specific integer, floating-point, Boolean, or list evaluation; LogicBench normalizes yes/no answers; MedCalcBench uses its numeric tolerance rules; and CHAMP uses its numeric or closed-form evaluator. Verifier exceptions are treated as incorrect predictions.

B PROMPT TEMPLATES

This section provides the prompt templates used by FIRE. Placeholders enclosed in braces are instantiated at runtime. For all tasks, the candidate catalog contains 16 skills; only skill metadata is exposed during selection, while skill contents are revealed after selection.

B.1 SELECTION AND EXECUTION

Skill Selection

```
You are a skill-augmented problem-solving agent. You may solve the
    problem
with the help of {n} external skill(s) from the catalog below.

## Candidate Skill Catalog (16 skills, contents hidden)
- [skill_xxxxx] Name: Description
- ... (16 entries)

## Problem
{question}

## Your task
Select exactly {n} skill(s) that provide the capabilities needed to
    solve
the problem. Judge only from the catalog entries.

Think briefly, then output one line per selected skill:
<select>skill_xxxxx</select>

If {n} > 1, output all selected skills and then close with:
<stop_select/>
```

Initial Execution

```
## Loaded Skill(s)
### Skill 1: [skill_xxxxx] Name
{skill body, truncated to 8000 chars}
...

## Problem
{question}

## Your task (step 2 of 2)
Use the loaded skill(s), or your own knowledge if they prove unhelpful
    .
Reason step by step, then:
<answer>...</answer>
{answer-format hint}
```

B.2 REFLECTION AND RECOVERY

When the initial execution fails, FIRE invokes *Reflect* to diagnose the failure and choose whether to retain or replace the current skill configuration. The resulting diagnosis and recovery strategy are then conditioned into the next invocation of *Execute*.

Skill Reflection

```
## Problem
{question}

## Loaded skill(s) (round 1)
{selected skills}

## Your previous solution (incorrect)
{previous solution}
```

```
Your answer "{previous answer}" is incorrect.
You have ONE recovery opportunity.

## Candidate Skill Catalog (unchanged)
- [skill_xxxxx] Name: Description
- ... (16 entries)

## Your task: Diagnose and Recover
1. DIAGNOSE the specific error. Identify the exact step or calculation
     that
   went wrong and determine whether the failure comes from skill
    selection,
   skill execution, or both.
2. STRATEGY: state one or two concrete changes for the next attempt.
3. RECONFIGURE the skill set according to your diagnosis.

If the current skill(s) are appropriate, keep them unchanged.
Otherwise, replace the inappropriate skill(s) with better candidates.
The final configuration must contain exactly {n} distinct catalog
    skills.

Output in exactly this format:
<reflect>
Attribution: selection|execution|mixed
Diagnosis: [quote the specific error]
Strategy: [what to do differently]
</reflect>
<reconfig>KEEP</reconfig>

or, when changing the skill configuration:
<reconfig>SET skill_xxxxx[, skill_yyyyy, ...]</reconfig>
```

Recovery Execution

```
## Loaded Skill(s)
### Skill 1: [skill_xxxxx] Name
{skill body}
...

## Problem
{question}

## Previous Attempt (Incorrect)
Your answer was: "{previous answer, <=200 chars}"
Your reasoning (truncated):
{previous solution, <=1500 chars}

## Your Diagnosis (from reflection)
{diagnosis, <=500 chars}

## Your Recovery Strategy
{strategy, <=500 chars}

## Recovery Task
Based on your diagnosis and strategy above, solve the problem
    correctly this
time. Avoid the specific error you identified. Reason step by step,
    then:
<answer>...</answer>
{answer-format hint}
```

B.3 CASE-MEMORY INJECTION

Case memory is inserted directly into the corresponding interaction prompt. Routing memory is exposed before selection, whereas execution memory is retrieved only after a skill has been selected.

Routing Memory

```
## Retrieved Case Memory (evidence, not ground truth)
Similar past cases suggest the following skill distinctions.
Do not copy answers from them.

- Similar {data_source}: required [{gold skill ids}] {skill name};
  task: {question, <=220 chars}

- Similar {data_source}: required [{gold skill ids}] {skill name};
  task: {question, <=220 chars}

- Similar but different case: required [{other skill ids}] {skill
    names};
  task: {question, <=180 chars}
```

At most two positive cases and one contrasting case are exposed to the selection policy.

Execution Memory

```
## Successful Usage Cases for [{skill_gid}]
These are sanitized patterns from related tasks.
Do not copy their final answers.

### Related task: {question, <=220 chars}
Pattern: {successful strategy or excerpt, <=650 chars}
Previous pitfall: {failure diagnosis, <=320 chars}
```

At most two execution cases are retrieved. All explicit `<answer>...</answer>` spans are removed from prompt-facing memory.

B.4 SKILL-REVISION TEMPLATES

Routing and execution failures modify different fields of a skill. The original skill is preserved, and all revisions are maintained as overlays.

Description Revision

```
[Selection hint: verified correct for {data_source} problems like:
"{question, <=120 chars}"]
```

Content Revision

```
### Usage Insights (learned from training experience)

**Common pitfalls to avoid:**
- {pitfall from failure diagnoses, up to 4}

**Recommended approaches:**
- {approach from recovery strategies, up to 4}
```

At most eight recent insight records are retained for each skill, and a description hint is appended at most once per skill.

Answer-format constraints. Execution prompts additionally specify the dataset-specific output format: TheoremQA uses integer, decimal, Boolean, or list answers as appropriate; LogicBench requires `yes`/`no`; MedCalcBench requires a numeric value; and CHAMP accepts a number or closed-form expression. All final predictions are enclosed by `<answer>...</answer>`.

C Case Study: From Failure to Persistent Improvement

We present a TheoremQA example that traces a routing miss through reflective recovery, case-memory construction, and persistent skill revision.

C.1 Case Overview

Task and Skill Configuration

Task	Find the sum of all positive integers less than 196 and relatively prime to 98.
Gold skill	`skill_00230`: *Euler's Totient Theorem and Phi Function*.
Initial skill	`skill_00387`: *Divisibility from a Product Divisor*.
Catalog	16 candidate skills; gold-set cardinality $\lvert\mathcal{G}_x\rvert = 1$.

At training dump 49, none of the eight rollouts initially selects the gold skill: seven choose `skill_00387` and one chooses `skill_00386`. The group therefore provides a clear routing-miss example for examining failure recovery and subsequent library updates.

C.2 Rollout-Group Outcomes

Eight Rollouts from the Same Task

Rollouts	8	Initial gold hits	0
Initial `00387`	7	Final successes	8
Initial `00386`	1	Recovery successes	1

All eight initial selections miss the annotated gold skill. Among these trajectories, one exhibits the complete reflective recovery path: an incorrect first answer is followed by Replace to the gold skill and a correct second execution.

C.3 Successful Recovery Trajectory

Select → Execute → Reflect → Execute

1. Select. The agent initially chooses the non-gold skill `skill_00387`:

`<select>skill_00387</select>`.

2. Execute. The first execution applies the standard totient-sum formula in the wrong scope: it uses a result for numbers below a modulus when the task instead asks for numbers below 196 that are relatively prime to 98. This produces

`<answer>8037</answer>`,

which is rejected by the verifier; the ground-truth answer is 8232.

3. Reflect. The reflection identifies this misuse of the formula and proposes exploiting the period-98 structure of coprimality. It then replaces the initial skill with the gold skill:

`<reconfig>REPLACE skill_00230</reconfig>`.

The corresponding reflection contains:

> **Attribution:** execution
>
> **Diagnosis:** the formula $\frac{1}{2}N\varphi(N)$ was applied in the wrong context: the task sums numbers below 196 that are relatively prime to 98, rather than numbers below 98.
>
> **Strategy:** exploit the period of 98, compute the sum over one coprime residue block, and then account for the second block.

4. Execute again. With `skill_00230` loaded, the second execution follows this strategy. Since $\varphi(98) = 42$, the coprime residues below 98 sum to 2058. The corresponding residues in the second block sum to

$$2058 + 98 \times 42 = 6174,$$

giving

$$2058 + 6174 = 8232.$$

The agent returns

`<answer>8232</answer>`,

which is accepted by the verifier.

C.4 Persistent Update from the Same Failure

The recovered trajectory is not used only for the current episode. The same failure group also produces persistent updates to the Skill Library and Case Library.

Skill-Library Update

The successful replacement triggers the `P3_replace_recovery` pattern: the initial selection misses the gold skill, REPLACE switches to `skill_00230`, and the subsequent execution succeeds.
FIRE records this evidence as a description-level revision for `skill_00230`. The original skill remains unchanged, while the revision is stored as an overlay that can be used in subsequent interactions.

Case-Library Update

The same eight-rollout group is compressed into a single case card:

Episodes	8	Initial gold hits	0
Final successes	8	Recovery successes	1
Gold-selected successes	0	Success rate	1.0

Because no initial rollout selects the gold skill, the case contains no gold-selected successful execution strategy. Accordingly, FIRE leaves the corresponding successful-strategy field empty while retaining the routing statistics, failure evidence, and recovery record.
The case first enters the pending buffer and is published at the next library-update boundary, preventing it from being reused by the rollouts from which it was constructed.

Cross-Task Reuse

After publication, the resulting case is retrieved in 20 later interactions from other tasks. For example, it is retrieved for another TheoremQA task associated with `skill_00230`, showing that failure-derived experience is available for reuse beyond the original training instance.

Overall, this example traces how FIRE turns a routing miss into reusable supervision: reflection changes the skill configuration and corrects the answer from 8037 to 8232, while the resulting failure evidence is retained through persistent skill and case-memory updates for subsequent interactions.