SKILLFORGE: Co-Evolving Skills and Agents via Dynamic Skill Lifecycles

Yuyao Ge[1,2,3] **Yiwei Wang**[4] **Yuchen He**[1,2,3] **Baolong Bi**[1,2,3]
Lingrui Mei[1,2,3] **Jiayu Yao**[1,2,3] **Lizhe Chen**[5] **Shenghua Liu**[1,2,3*]

[1]State Key Laboratory of AI Safety
[2]Institute of Computing Technology, Chinese Academy of Sciences
[3]University of Chinese Academy of Sciences
[4]University of California, Merced [5]Tsinghua University
{geyuyao24z, liushenghua}@ict.ac.cn

https://github.com/YuyaoGe/SkillForge

Abstract

Memory-augmented reinforcement learning strengthens LLM agents' ability to solve complex long-horizon tasks. Skills are one such form of memory, pairing instructions with an applicability condition over task types. However, retaining every skill indiscriminately as the policy improves lets obsolete or harmful entries accumulate and mislead the agent. We propose **SKILLFORGE**, an agentic RL method that compiles and evolves the skill library through a fitness-driven skill lifecycle of `trial`, `active`, `stable`, and `retired` states, so that the skills and the model co-evolve throughout training. A pre-RL evaluation phase first uses the base model's own rollouts to pre-retire low-fitness skills, yielding a filtered library that then seeds supervised fine-tuning. Reinforcement learning takes over from this checkpoint, and at each iteration selective retirement, stabilization, and LLM-guided mutation continue to forge the skill library alongside policy optimization. Across multiple interactive agent benchmarks, SKILLFORGE achieves the highest aggregate success rate, delivering up to 7.8% relative improvement over the strongest baseline while keeping the skill library compact throughout training. We construct **SKILLFURNACE**, a dataset of 5k+ annotated records bundling retirement-filtered SFT trajectories, evolved skill libraries with fitness annotations, and retirement events with human-annotated failure categories.

1 Introduction

Generalist agents such as Claude Code [1] and OpenClaw [2] derive much of their capability from modular, reusable abstractions provided at inference time. Even frontier large language models (LLMs) sit behind a fixed knowledge cutoff, so they cannot absorb personalized workflows [3] or adapt to post-deployment tooling conventions [4]. To close this gap at inference time, recent work centers on the *skill*: a bundle of instructions, examples, and applicability conditions retrieved into the agent's context on demand. Skill libraries are assembled from procedural memory combining step-level and script-level abstractions [5], or through autonomous skill-discovery loops [6]. As libraries grow, however, retrieval-based augmentation degrades: top-k retrieval returns noisier context, outdated skills crowd out high-quality ones, and many entries no longer match the agent's current

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

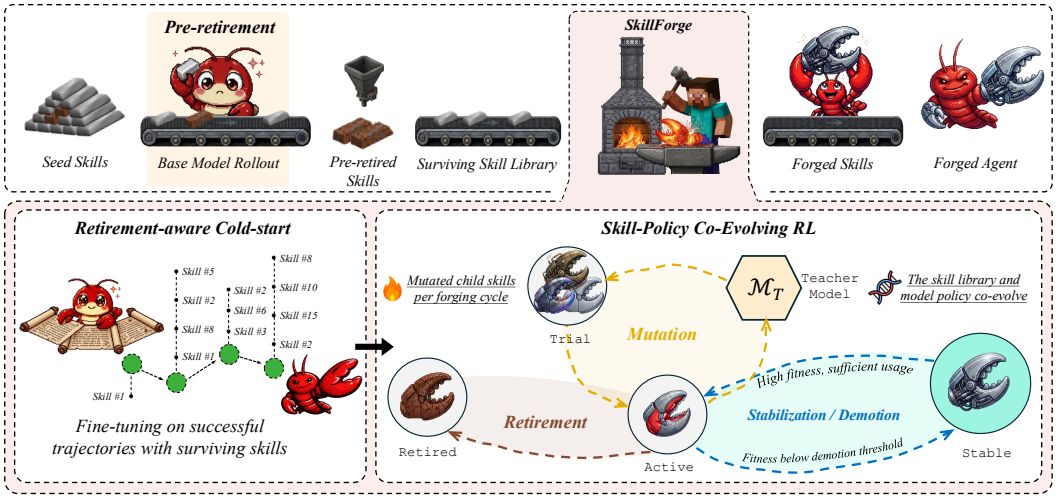


Figure 1: Overview of SKILLFORGE. **Top**: seed skills are pre-retired under the base model's own rollouts; surviving fitness-verified skills seed retirement-aware cold-start and skill-policy co-evolving RL, yielding a trained agent with an adapted skill library. **Bottom-left**: cold-start fine-tunes on successful trajectories with surviving skills. **Bottom-right**: skills and model co-evolve through the fitness-driven lifecycle across `trial`, `active`, `stable`, and `retired` via selective retirement, stabilization, demotion, and LLM-guided mutation.

capability. Li et al. [7] confirm these risks empirically: curated libraries contain skills that hurt performance, and larger libraries yield diminishing returns.

A complementary line of work trains the policy via agentic reinforcement learning (Agentic RL) against environment reward. Pairing skills or other accumulated memory with RL is a natural combination: skills focus exploration with behavioral priors, while RL shapes the policy through interaction. Recent skill-augmented RL injects retrieved skills into the agent's context each rollout [8]; related threads close an offline-to-online distillation loop [9] or run policy updates directly over episodic memory [10]. Yet these methods largely treat the library as append-only, leaving the agent's context polluted by low-quality skills, bloated by unbounded accumulation, and misled by once-useful skills that turn counterproductive as the policy improves, a failure mode we call *delayed obsolescence*: a skill whose fitness once reached the stable state but later drops below the retirement threshold as the policy matures.

We propose SKILLFORGE, which treats skill library management as an active *forging* process rather than passive accumulation. Starting from a seed skill library distilled from trajectories, as illustrated in Figure 1, SKILLFORGE runs in three stages: **(i)** pre-retire seed skills that fail under the base model's own rollouts, **(ii)** cold-start fine-tuning on the surviving fitness-verified skills, and **(iii)** switch to RL training where skills and model co-evolve each iteration. Each cycle reads runtime fitness and moves every skill along the skill lifecycle `trial`, `active`, `stable`, `retired`, with borderline-fitness skills rewritten by an LLM-guided mutation operator. Skills and model co-evolve, keeping the active set compact and aligned with the agent's capability. We contribute the following:

- **SKILLFORGE and the skill lifecycle**: We propose SKILLFORGE, a fitness-driven skill lifecycle augmented with LLM-guided mutation that moves each skill through `trial`, `active`, `stable`, and `retired` states so that skills and model co-evolve, replacing the append-only accumulation of prior skill-augmented RL.
- **SKILLFURNACE dataset**: We construct SKILLFURNACE, a three-part bundle of 5k+ annotated records spanning retirement-filtered SFT trajectories, evolved skill libraries with fitness annotations, and retirement events with human-annotated failure categories.
- **Empirical validation**: Across three interactive agent benchmarks spanning embodied control, web navigation, and Search-Augmented QA, SKILLFORGE surpasses the strongest baseline on every environment while keeping the library compact, delivering up to 7.8% relative improvement in aggregate success rate.

2 Related Work

Experience accumulation for LLM agents. Early work took the form of in-context reflection: self-critiquing the current output [11, 12], and subsequently mining cross-trajectory insights [13, 14]. In parallel, skill-library methods emerged: Wang et al. [15] introduced an executable-code library for open-ended exploration, followed by reusable task workflows [16], and more recently procedural memory mixing step-level instructions with script-level abstractions [5]. A related thread autonomously discovers new skills for the agent to practice [6, 17]. On the RL side, memory-augmented methods couple such repositories to the training loop: earlier work used external memory to support tool-assisted planning [18, 19], and later closed an offline-to-online loop between distillation and retrieval [9]. Most recently, Xia et al. [8] combined skill libraries with RL training, injecting retrieved skills into the context during each rollout. In most of them the repository is effectively monotone: entries persist once added and cannot be corrected as the policy evolves. SKILLFORGE instead treats the library as a dynamic population under fitness-driven selection, retiring low-fitness skills and reshaping borderline ones via LLM-guided mutation in step with the online RL optimizer.

LLM-guided evolutionary optimization. Jaderberg et al. [20] introduced population-based training, applying fitness-based selection and mutation over the hyperparameter schedules of concurrent workers. The same evaluate-select-mutate template has since been instantiated with LLMs: LLM-as-optimizer rewrites evolve a population of natural-language instructions against a fixed held-out metric [21, 22], self-referential variants let instructions evolve each other [23], and quality-diversity searches pair a pretrained LLM with an evaluator to discover diverse high-performing programs [24, 25]. These methods all optimize static text artifacts against a stationary held-out metric in an offline loop. SKILLFORGE brings fitness-driven selection and mutation into LLM agentic reinforcement learning, where the evolving entities are *skills* that augment the agent's context during interactive rollouts. Joint online training makes skill fitness inherently non-stationary, requiring a lifecycle that retires and reshapes skills as the policy evolves.

3 Preliminaries

3.1 Group Relative Policy Optimization

GRPO [26] estimates advantages from grouped samples without a separate value function. Given task description d drawn from a training task distribution $\mathcal{D}_{\text{task}}$, a uniform distribution over the training tasks of each environment, the old policy $\pi_{\theta_{\text{old}}}$ samples G episodes $\{\tau_n\}_{n=1}^{G}$ with binary rewards $r_n = R(\tau_n) \in \{0, 1\}$, and computes group-normalized advantages $\hat{A}_n = (r_n - \mu_G)/\sigma_G$. Let $\boldsymbol{y}_n = (y_{n,1}, \ldots, y_{n,|\boldsymbol{y}_n|})$ denote the concatenation of action tokens produced autoregressively by the policy during episode τ_n; agent contexts and observations serve as conditioning inputs rather than generation targets, and $\boldsymbol{y}_n$ aligns with the step-level actions $\{a_t\}$ of Section 3.2. Token-level importance ratios are $\rho_{n,l} = \pi_\theta(y_{n,l} \mid c_{n,l})/\pi_{\theta_{\text{old}}}(y_{n,l} \mid c_{n,l})$, where $c_{n,l}$ denotes the left context of token $y_{n,l}$, comprising all prior agent contexts, observations, and earlier action tokens of the episode. The index n indexes episodes within a GRPO group and is distinct from the skill index i used in Section 4. The policy is updated by maximizing:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{d\sim\mathcal{D}_{\text{task}},\,\{\tau_n\}}\left[\frac{1}{G}\sum_{n=1}^{G}\frac{1}{|\boldsymbol{y}_n|}\sum_{l=1}^{|\boldsymbol{y}_n|}\min\big(\rho_{n,l}\,\hat{A}_n,\ \text{clip}(\rho_{n,l},\,1-\epsilon,\,1+\epsilon)\,\hat{A}_n\big)\right] - \beta\, D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{1}$$

with token-level clipping and a trajectory-level KL penalty against the reference policy.

3.2 Skill-Augmented RL

SkillRL [8] augments GRPO with a textual skill library $\mathcal{S}$, where each skill $s = (v, b, \omega)$ comprises a name v, a behavioral principle b, and an applicability condition ω. Given task description d from Eq. (1), a retrieval function $\phi(d, \mathcal{S})$ selects the top-K skills whose condition ω matches the task-type label of d and concatenates them into the agent's context:

$$p_t = d \oplus \phi(d, \mathcal{S}) \oplus h_t \oplus o_t, \tag{2}$$

where h_t is a sliding window of the last W observation–action pairs, o_t is the current observation, and $\oplus$ denotes string concatenation. At each step t, an action $a_t \sim \pi_\theta(\cdot \mid p_t)$ is sampled, and the trajectory $\tau = (p_1, a_1, o_2, \ldots, p_T, a_T)$ decomposes into context–action pairs $\{(p_t, a_t)\}_{t=1}^{T}$ used as SFT supervision units. Each episode receives a binary reward $R(\tau) \in \{0, 1\}$. Initially, skills are distilled from trajectories via OpenAI o3 [27], and a cold-start fine-tuning checkpoint $\pi_{\theta_{\text{sft}}}$ serves as both RL initialization and reference policy π_{ref}.

4 SKILLFORGE

As illustrated in Figure 1, SKILLFORGE runs in three stages: **(i)** pre-retirement evaluation filters low-fitness seed skills, **(ii)** retirement-aware cold-start fine-tuning on the surviving library, and **(iii)** agentic RL with a fitness-driven skill lifecycle where skills and policy co-evolve each iteration. This design addresses the non-stationarity of skill fitness under online RL: as the policy matures, once-useful skills may slip below the retirement threshold, *delayed obsolescence* formalized in Eq. (11).

4.1 SKILLFURNACE Initialization

We evaluate the initial skill library $\mathcal{S}_0$ released by SkillRL [8] with base model π_{θ_0}, which performs M rollout episodes with skills injected. For each $s_i \in \mathcal{S}_0$ we accumulate usage $U_i^{(0)}$ and success $C_i^{(0)}$. Skills whose proto-fitness, their success rate under the base model's own rollouts, falls below a conservative threshold δ_{pre} and that have been used at least $N_{\min}$ times are pre-retired:

$$\hat{f}_i = \frac{C_i^{(0)}}{U_i^{(0)}}, \qquad \mathcal{S}_{\text{retire}}^{(0)} = \{s_i \in \mathcal{S}_0 \mid \hat{f}_i < \delta_{\text{pre}} \wedge U_i^{(0)} \geq N_{\min}\}. \tag{3}$$

The surviving library $\mathcal{S}^{(0)} = \mathcal{S}_0 \setminus \mathcal{S}_{\text{retire}}^{(0)}$ seeds both the cold-start fine-tuning data and the RL phase described in Section 4.4. Let $\mathcal{T}_{\text{eval}}$ denote the trajectories from this evaluation, and $\phi_{\text{eval}}(d, \mathcal{S}_0)$ the retrieval ϕ from Eq. (2) on the full library $\mathcal{S}_0$ before pre-retirement. We construct the SFT dataset from successful trajectories:

$$\mathcal{D}_{\text{SFT}} = \{(p_t, a_t) \mid \tau \in \mathcal{T}_{\text{eval}},\ (p_t, a_t) \in \tau,\ R(\tau) = 1,\ \phi_{\text{eval}}(d_\tau, \mathcal{S}_0) \cap \mathcal{S}_{\text{retire}}^{(0)} = \emptyset\}, \tag{4}$$

where d_τ is the task description of τ and $R(\tau)$ its binary reward; the filter excludes trajectories that relied on subsequently retired skills. Fine-tuning the base model on $\mathcal{D}_{\text{SFT}}$ via standard cross-entropy

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(p_t, a_t) \sim \mathcal{D}_{\text{SFT}}} \left[\log \pi_\theta(a_t \mid p_t)\right], \tag{5}$$

produces $\pi_{\theta_{\text{sft}}}$, which serves as both the RL initialization and the reference policy π_{ref}.

The filtered $\mathcal{D}_{\text{SFT}}$, the surviving library $\mathcal{S}^{(0)}$, and human-annotated retirement events from previously recorded training runs together form our SKILLFURNACE dataset. As summarized in Table 1, retired skills peak at an average fitness of 0.67, close to the stabilization threshold δ_{stable} of 0.7, before dropping by 0.28 at retirement, and 57.2% first reach the `stable` state before this decay: a direct empirical signature of *delayed obsolescence*. Appendix C gives per-component details.

4.2 Skill Fitness Tracking

For each skill s_i, we maintain a usage count U_i (episodes in which s_i was retrieved) and a success count C_i (episodes that succeeded). At the start of RL, the counters are initialized from the preliminary evaluation phase: $C_i \leftarrow C_i^{(0)}$ and $U_i \leftarrow U_i^{(0)}$ for each surviving skill $s_i \in \mathcal{S}^{(0)}$, so proto-fitness information from Section 4.1 is preserved rather than discarded. The fitness is:

$$f_i = \begin{cases} C_i / U_i & \text{if } U_i \geq N_{\text{warm}} \\ f_{\text{default}} & \text{otherwise} \end{cases} \tag{6}$$

where N_{warm} is a minimum usage count before trusting the empirical success rate and f_{default} is a neutral prior for new skills. We use $f_i^{(k)}$ for the running estimate at training step k when dependence on k matters, as formalized in Section 4.4; for any skill with $U_i^{(0)} \geq N_{\text{warm}}$ we have $f_i^{(0)} = \hat{f}_i$, so the proto-fitness from Eq. (3) is the runtime fitness at step 0, reused rather than recomputed.

Table 1: SKILLFURNACE dataset statistics. The top block reports per-component record counts across SFT trajectories, library snapshots, and retirement events; the bottom block reports detailed retirement-event metrics. **Total records (#)** is summed across components within each environment.

Statistic	ALFWorld	WebShop	Search-Aug. QA	Total
Dataset scale				
SFT trajectories (#)	412	348	253	1,013
Library snapshot records (#)	1,512	1,683	1,326	4,521
Retirement events (#)	138	121	59	318
Total records (#)	**2,062**	**2,152**	**1,638**	**5,852**
Retirement event statistics				
Avg. peak fitness	0.696	0.709	0.515	0.667
Avg. fitness drop from peak	0.294	0.279	0.239	0.278
Avg. lifespan (steps)	38.8	31.7	49.4	38.0
Stable before retirement (%)	60.1	58.7	47.5	57.2

Credit is assigned at the episode level: for each episode with trajectory τ and retrieved skill set $\mathcal{S}_\tau = \phi(d, \mathcal{S})$, all K retrieved skills receive the binary reward signal:

$$U_i \leftarrow U_i + \mathbb{1}[s_i \in \mathcal{S}_\tau], \quad C_i \leftarrow C_i + \mathbb{1}[s_i \in \mathcal{S}_\tau] \cdot R(\tau), \tag{7}$$

where $\mathbb{1}[\cdot]$ denotes the indicator function.

4.3 Skill Lifecycle Management

Based on accumulated fitness, we define a *lifecycle update rule* ℓ that assigns each skill one of four states $\{\texttt{trial}, \texttt{active}, \texttt{stable}, \texttt{retired}\}$; the active library $\mathcal{S}$ retains only non-retired skills. Transitions depend on fitness thresholds $\delta_{\text{retire}} < \delta_{\text{demote}} < \delta_{\text{stable}}$, where δ_{demote} lies strictly between the retirement and stabilization thresholds to provide hysteresis so that a stable skill whose fitness slips must first be demoted to `active` before becoming eligible for retirement. Transitions also depend on a mutation range $[\delta_{\text{low}}, \delta_{\text{high}}]$, minimum usage counts N_{promote} and N_{stable}, and a generation-aware retirement threshold:

$$N_{\text{retire}}^{(g_i)} = \begin{cases} N_{\text{protect}} & \text{if } g_i = 0 \\ N_{\text{retire}} & \text{otherwise} \end{cases} \tag{8}$$

where g_i is the mutation generation; distilled ($g_i = 0$) skills require a higher usage bar $N_{\text{protect}} > N_{\text{retire}}$ before they can be retired. The lifecycle update rule is then:

$$\ell(s_i) \leftarrow \begin{cases} \texttt{active} & \text{if } \ell(s_i) = \texttt{trial} \wedge U_i \geq N_{\text{promote}} \\ \texttt{stable} & \text{if } \ell(s_i) = \texttt{active} \wedge f_i \geq \delta_{\text{stable}} \wedge U_i \geq N_{\text{stable}} \\ \texttt{active} & \text{if } \ell(s_i) = \texttt{stable} \wedge f_i < \delta_{\text{demote}} \\ \texttt{retired} & \text{if } \ell(s_i) = \texttt{active} \wedge f_i < \delta_{\text{retire}} \wedge U_i \geq N_{\text{retire}}^{(g_i)} \end{cases} \tag{9}$$

If none applies, for example an active skill with $\delta_{\text{retire}} \leq f_i < \delta_{\text{stable}}$, $\ell(s_i)$ is left unchanged; Appendix B states the well-definedness and per-cycle ordering used by Algorithm 1. Only `active` skills are eligible for retirement; `trial` skills must first accumulate N_{promote} usages and promote to `active`, and `stable` skills must first be demoted. This two-gate design prevents premature retirement of newly created skills and shields proven skills from transient fitness dips. The mutation generation g_i is 0 for skills distilled into $\mathcal{S}_0$ and is incremented by 1 at each mutation.

Initial skills surviving pre-retirement enter as `stable` if $\hat{f}_i \geq \delta_{\text{stable}}$, or as `active` otherwise; newly generated skills enter as `trial`. The lifecycle operates through promotion, demotion, retirement, stabilization, and mutation, per Algorithm 1. Because demotion precedes retirement, a skill can be demoted and retired in one cycle, so once-stable skills can be retired as the policy evolves.

Mutation reshapes `active` skills whose fitness lies in a moderate range $[\delta_{\text{low}}, \delta_{\text{high}}]$ and which have been used at least N_{mutate} times, so that the fitness estimate used to select mutation parents is informative. From the resulting candidate pool $\mathcal{M}_{\text{cand}}$, we select up to K_{mutate} parents via inverse-fitness-weighted sampling ($P(s_i) \propto 1 - f_i$), collect their recent failure trajectories $\mathcal{T}_{\text{fail}}$, and query our teacher model ($\mathcal{M}_T$) to generate improved variants:

$$\begin{aligned} \mathcal{M}_{\text{cand}} &= \{s_i \in \mathcal{S} \mid \ell(s_i) = \texttt{active} \wedge \delta_{\text{low}} \leq f_i \leq \delta_{\text{high}} \wedge U_i \geq N_{\text{mutate}}\}, \\ s_{\text{child}} &= \mathcal{M}_T\big(\mathcal{P}_\mu(s_{\text{parent}},\, f_{\text{parent}},\, \mathcal{T}_{\text{fail}})\big), \end{aligned} \tag{10}$$

Algorithm 1 SKILLFORGE Forging Cycle

Require: Skill library $\mathcal{S}$, fitness scores $\{f_i\}$, usage counts $\{U_i\}$, failed trajectories $\mathcal{T}_{\text{fail}}$
Ensure: Updated skill library $\mathcal{S}$

for $s_i \in \mathcal{S}$ where $\ell(s_i) = \texttt{trial} \wedge U_i \geq N_{\text{promote}}$ **do**
  $\ell(s_i) \leftarrow \texttt{active}$ ▷ Promote: trial → active
end for
for $s_i \in \mathcal{S}$ where $\ell(s_i) = \texttt{stable} \wedge f_i < \delta_{\text{demote}}$ **do**
  $\ell(s_i) \leftarrow \texttt{active}$ ▷ Demote: stable → active
end for
$\mathcal{R} \leftarrow \{s_i \in \mathcal{S} \mid \ell(s_i) = \texttt{active} \wedge f_i < \delta_{\text{retire}} \wedge U_i \geq N_{\text{retire}}^{(g_i)}\}$ ▷ Retirement candidates
$\mathcal{S} \leftarrow \mathcal{S} \setminus \text{bottom-}m(\mathcal{R}),\ m = \min(K_{\text{retire}}, |\mathcal{R}|)$ ▷ bottom-$m(\mathcal{R})$: m skills in $\mathcal{R}$ with lowest f_i
for $s_i \in \mathcal{S}$ where $\ell(s_i) = \texttt{active} \wedge f_i \geq \delta_{\text{stable}} \wedge U_i \geq N_{\text{stable}}$ **do**
  $\ell(s_i) \leftarrow \texttt{stable}$ ▷ Stabilize: active → stable
end for
$\mathcal{M}_{\text{cand}} \leftarrow \{s_i \in \mathcal{S} \mid \ell(s_i) = \texttt{active} \wedge \delta_{\text{low}} \leq f_i \leq \delta_{\text{high}} \wedge U_i \geq N_{\text{mutate}}\}$ ▷ Mutation pool
Select K_{mutate} parents from $\mathcal{M}_{\text{cand}}$; $P(s_i) \propto 1 - f_i$ ▷ Lower fitness → higher chance
for each selected s_{parent} **do**
  $s_{\text{child}} \leftarrow \mathcal{M}_T(\mathcal{P}_\mu(s_{\text{parent}}, f_{\text{parent}}, \mathcal{T}_{\text{fail}}))$ ▷ LLM-guided mutation
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_{\text{child}}\}$; $\ell(s_{\text{child}}) \leftarrow \texttt{trial}$; $g_{\text{child}} \leftarrow g_{\text{parent}} + 1$ ▷ Child enters as trial
end for
while $|\mathcal{S}| > S_{\max}$ **and** $\exists s_i \in \mathcal{S}$ with $\ell(s_i) = \texttt{active}$ **do**
  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\arg\min_{s_i:\, \ell(s_i)=\texttt{active}} f_i\}$ ▷ Enforce size bound
end while
return $\mathcal{S}$

where $\mathcal{P}_\mu$ assembles the parent skill's content, fitness statistics, and failure examples into a structured template. Each child enters with $\ell(s_{\text{child}}) = \texttt{trial}$ and generation $g_{\text{child}} = g_{\text{parent}} + 1$. The inverse-fitness weighting $P(s_i) \propto 1 - f_i$ biases selection toward skills with more room to improve without concentrating mutation on a single outlier. We instantiate $\mathcal{M}_T$ from a different model family than the trained policy, since Li et al. [7] report that self-generated skills provide negligible or negative benefit compared with externally authored ones.

4.4 Training Integration

As given in Algorithm 1, a forging cycle runs every E validation epochs, with per-cycle budgets K_{retire} and K_{mutate} and a total skill cap $S_{\max}$ bounding computational cost.

Making the non-stationarity introduced at the start of Section 4 explicit, the fitness estimate at training step k depends on the policy trajectory:

$$f_i^{(k)} = \frac{\sum_{j \leq k} \mathbb{1}[s_i \in \mathcal{S}_{\tau^{(j)}}] \cdot R(\tau^{(j)})}{\sum_{j \leq k} \mathbb{1}[s_i \in \mathcal{S}_{\tau^{(j)}}]}, \quad \tau^{(j)} \sim \pi_{\theta_j}(\cdot \mid p^{(j)}), \tag{11}$$

where $p^{(j)}$ denotes the agent's context at step j, constructed via Eq. (2) with the library $\mathcal{S}^{(j)}$. The $j = 0$ term covers the pre-retirement rollouts under π_{θ_0}, consistent with the counter initialization in Section 4.2 and yielding $f_i^{(0)} = \hat{f}_i$. As Li et al. [28] report, on-policy formulations avoid the exposure bias of off-policy alternatives, justifying the non-stationarity above. At each step k, the policy is updated via Eq. (1) conditioned on the current library, and the forging cycle updates the library based on accumulated fitness:

$$\theta_{k+1} = \arg\max_\theta \mathcal{J}_{\text{GRPO}}(\theta;\, \mathcal{S}^{(k)}), \qquad \mathcal{S}^{(k+1)} = \text{Forge}\left(\mathcal{S}^{(k)},\, \{f_i^{(k)}\}\right), \tag{12}$$

where Forge denotes a full forging cycle. Skills and policy thus co-evolve: as θ improves, fitness estimates shift, and the forging cycle retires skills whose utility has diminished. *Delayed obsolescence* is then the formal event $\exists k_1 < k_2 :\ f_i^{(k_1)} \geq \delta_{\text{stable}} \wedge f_i^{(k_2)} < \delta_{\text{retire}}$.

Table 2: Performance on ALFWorld and WebShop (%). WebShop **Score** is the partial-credit reward; **Succ.** is binary task success. **Bold**: best; underline: second best.

Method	ALFWorld							WebShop	
	Pick	Look	Clean	Heat	Cool	Pick2	All	Score	Succ.
Closed-source LLMs									
GPT-4o	75.3	60.8	31.2	56.7	21.6	49.8	48.0	31.8	23.7
Gemini-2.5-Pro	92.8	63.3	62.1	69.0	26.6	58.7	60.3	42.5	35.9
Base Model									
Vanilla	33.4	21.6	19.3	6.9	2.8	3.2	14.8	26.4	7.8
Context / Memory Methods									
ReAct	$48.5_{\blacktriangle 15.1}$	$35.4_{\blacktriangle 13.8}$	$34.3_{\blacktriangle 15.0}$	$13.2_{\blacktriangle 6.3}$	$18.2_{\blacktriangle 15.4}$	$17.6_{\blacktriangle 14.4}$	$31.2_{\blacktriangle 16.4}$	$46.2_{\blacktriangle 19.8}$	$19.5_{\blacktriangle 11.7}$
Reflexion	$62.0_{\blacktriangle 28.6}$	$41.6_{\blacktriangle 20.0}$	$44.9_{\blacktriangle 25.6}$	$30.9_{\blacktriangle 24.0}$	$36.3_{\blacktriangle 33.5}$	$23.8_{\blacktriangle 20.6}$	$42.7_{\blacktriangle 27.9}$	$58.1_{\blacktriangle 31.7}$	$28.8_{\blacktriangle 21.0}$
Mem0	$54.0_{\blacktriangle 20.6}$	$55.0_{\blacktriangle 33.4}$	$26.9_{\blacktriangle 7.6}$	$36.4_{\blacktriangle 29.5}$	$20.8_{\blacktriangle 18.0}$	$7.7_{\blacktriangle 4.5}$	$33.6_{\blacktriangle 18.8}$	$23.9_{\blacktriangledown 2.5}$	$2.0_{\blacktriangledown 5.8}$
ExpeL	$21.0_{\blacktriangledown 12.4}$	$67.0_{\blacktriangle 45.4}$	$55.0_{\blacktriangle 35.7}$	$52.0_{\blacktriangle 45.1}$	$71.0_{\blacktriangle 68.2}$	$6.0_{\blacktriangle 2.8}$	$46.3_{\blacktriangle 31.5}$	$30.9_{\blacktriangle 4.5}$	$11.2_{\blacktriangle 3.4}$
MemP	$54.3_{\blacktriangle 20.9}$	$38.5_{\blacktriangle 16.9}$	$48.1_{\blacktriangle 28.8}$	$56.2_{\blacktriangle 49.3}$	$32.0_{\blacktriangle 29.2}$	$16.7_{\blacktriangle 13.5}$	$41.4_{\blacktriangle 26.6}$	$25.3_{\blacktriangledown 1.1}$	$6.4_{\blacktriangledown 1.4}$
SimpleMem	$64.5_{\blacktriangle 31.1}$	$33.3_{\blacktriangle 11.7}$	$20.0_{\blacktriangle 0.7}$	$12.5_{\blacktriangle 5.6}$	$33.3_{\blacktriangle 30.5}$	$3.8_{\blacktriangle 0.6}$	$29.7_{\blacktriangle 14.9}$	$33.2_{\blacktriangle 6.8}$	$8.6_{\blacktriangle 0.8}$
RL-based Methods									
RLOO	$87.6_{\blacktriangle 54.2}$	$\underline{78.2}_{\blacktriangle 56.6}$	$87.3_{\blacktriangle 68.0}$	$81.3_{\blacktriangle 74.4}$	$71.9_{\blacktriangle 69.1}$	$48.9_{\blacktriangle 45.7}$	$75.5_{\blacktriangle 60.7}$	$80.3_{\blacktriangle 53.9}$	$65.7_{\blacktriangle 57.9}$
GRPO	$90.8_{\blacktriangle 57.4}$	$66.1_{\blacktriangle 44.5}$	$89.3_{\blacktriangle 70.0}$	$74.7_{\blacktriangle 67.8}$	$72.5_{\blacktriangle 69.7}$	$64.7_{\blacktriangle 61.5}$	$77.6_{\blacktriangle 62.8}$	$79.3_{\blacktriangle 52.9}$	$66.1_{\blacktriangle 58.3}$
Memory-Augmented RL									
MemRL	$62.8_{\blacktriangle 29.4}$	$38.5_{\blacktriangle 16.9}$	$22.2_{\blacktriangle 2.9}$	$12.5_{\blacktriangle 5.6}$	$8.0_{\blacktriangle 5.2}$	$0.0_{\blacktriangledown 3.2}$	$21.4_{\blacktriangle 6.6}$	$29.5_{\blacktriangle 3.1}$	$9.2_{\blacktriangle 1.4}$
EvolveR	$64.9_{\blacktriangle 31.5}$	$33.3_{\blacktriangle 11.7}$	$46.4_{\blacktriangle 27.1}$	$13.3_{\blacktriangle 6.4}$	$33.3_{\blacktriangle 30.5}$	$33.3_{\blacktriangle 30.1}$	$43.8_{\blacktriangle 29.0}$	$42.5_{\blacktriangle 16.1}$	$17.6_{\blacktriangle 9.8}$
Mem0+GRPO	$78.1_{\blacktriangle 44.7}$	$54.8_{\blacktriangle 33.2}$	$56.1_{\blacktriangle 36.8}$	$31.0_{\blacktriangle 24.1}$	$65.0_{\blacktriangle 62.2}$	$26.9_{\blacktriangle 23.7}$	$54.7_{\blacktriangle 39.9}$	$58.1_{\blacktriangle 31.7}$	$37.5_{\blacktriangle 29.7}$
SimpleMem+GRPO	$89.5_{\blacktriangle 56.1}$	$36.3_{\blacktriangle 14.7}$	$60.0_{\blacktriangle 40.7}$	$50.0_{\blacktriangle 43.1}$	$64.9_{\blacktriangle 62.1}$	$26.3_{\blacktriangle 23.1}$	$62.5_{\blacktriangle 47.7}$	$67.8_{\blacktriangle 41.4}$	$46.9_{\blacktriangle 39.1}$
SkillRL	$\underline{97.9}_{\blacktriangle 64.5}$	$71.4_{\blacktriangle 49.8}$	$\mathbf{90.0}_{\blacktriangle 70.7}$	$\underline{90.0}_{\blacktriangle 83.1}$	$\underline{95.5}_{\blacktriangle 92.7}$	$\underline{87.5}_{\blacktriangle 84.3}$	$\underline{89.9}_{\blacktriangle 75.1}$	$\underline{85.2}_{\blacktriangle 58.8}$	$\underline{72.7}_{\blacktriangle 64.9}$
SKILLFORGE	$\mathbf{98.2}_{\blacktriangle 64.8}$	$\mathbf{86.6}_{\blacktriangle 65.0}$	$\underline{89.8}_{\blacktriangle 70.5}$	$\mathbf{93.7}_{\blacktriangle 86.8}$	$\mathbf{95.6}_{\blacktriangle 92.8}$	$\mathbf{88.4}_{\blacktriangle 85.2}$	$\mathbf{92.4}_{\blacktriangle 77.6}$	$\mathbf{87.8}_{\blacktriangle 61.4}$	$\mathbf{78.4}_{\blacktriangle 70.6}$

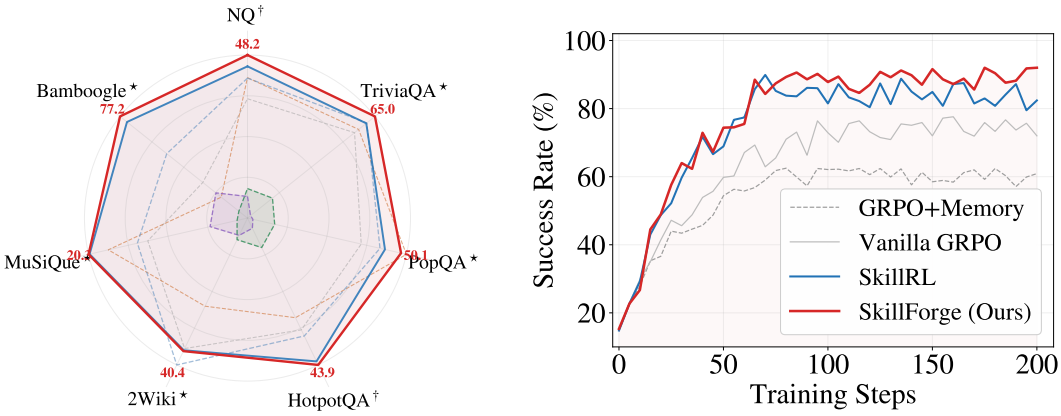
5 Experiments

5.1 Experimental Setup

Environments. Three interactive agent benchmarks are used: ALFWorld [29], WebShop [30], and Search-Augmented QA [31], with the Search-Augmented QA setting spanning NQ [32], TriviaQA [33], PopQA [34], HotpotQA [35], 2WikiMultiHopQA [36], MuSiQue [37], and Bamboogle [38].

Baselines. Baselines fall into three categories. **Context and memory methods** rely on in-context reasoning or external experience without updating the policy: ReAct [39], Reflexion [11], Mem0 [3], ExpeL [14], MemP [5], and SimpleMem [4]. **RL-based methods** optimize the policy via group-relative advantage estimation: RLOO [40] and GRPO [26]. **Memory-augmented RL methods** couple persistent memory with policy optimization and form the most direct comparison: MemRL [10], EvolveR [9], SkillRL [8], and two additional combinations, Mem0+GRPO and SimpleMem+GRPO, implemented by adding GRPO optimization to the respective memory mechanisms. For Search-Augmented QA the comparison additionally includes Search-o1 [41], R1-Instruct [42], Search-R1 [31], ZeroSearch [43], and StepSearch [44]. GPT-4o [45] and Gemini-2.5-Pro [46] are reported as frontier reference points.

Implementation. All experiments are built on the *verl* [47] framework with GRPO as the policy optimizer. The base model is Qwen2.5-7B-Instruct [48], first fine-tuned via the retirement-aware SFT described in Section 4.1 to produce environment-specific initializations. Training runs for 200 steps with a learning rate of 1×10^{-6}, clipping range $\epsilon = 0.2$, and a KL penalty coefficient $\beta = 0.001$. Rollout generation uses temperature 1.0 with a group size of $G = 8$. The forging cycle executes every 10 training steps, with up to $K_{\text{mutate}} = 5$ mutations and $K_{\text{retire}} = 3$ retirements per cycle; the teacher model for mutation is Kimi-K2.5 [49]. Full hyperparameters, hardware details, and forging thresholds are listed in Appendix B.



(a) Search-Augmented QA comparison (%).

(b) ALFWorld validation success rate across training.

Figure 2: Main-results diagnostics. (a) Per-dataset accuracy on Search-Augmented QA; per-axis numbers in bold red report SKILLFORGE accuracy on each dataset. —: SKILLFORGE (Ours); —: SkillRL; - -: EvolveR; - -: ZeroSearch; - -: Search-R1; - -: Search-o1; - -: R1-Instruct. †In-domain; ⋆out-of-domain. (b) SKILLFORGE achieves faster convergence and higher success rates than vanilla GRPO and memory-augmented RL.

Table 3: Ablation study of SKILLFORGE across three environments. "Library" reports the final non-retired skill count (coinciding with $|\mathcal{S}_0|$ for −forging/−mutation). Each row disables one forging stage: w/o forging removes the entire SKILLFORGE lifecycle, w/o pre-retirement keeps runtime stages but skips the pre-RL fitness filter, w/o retirement disables skill removal and size capping, and w/o mutation disables LLM-guided refinement. **Bold**: best.

Configuration	ALFWorld		WebShop		Search-Aug. QA	
	Library	Succ. (%)	Library	Succ. (%)	Library	Succ. (%)
SKILLFORGE	100	**92.4**	95	**78.4**	85	**48.7**
w/o forging	44	$89.8_{\blacktriangledown 2.6}$	66	$72.6_{\blacktriangledown 5.8}$	41	$46.3_{\blacktriangledown 2.4}$
w/o pre-retirement	100	$90.7_{\blacktriangledown 1.7}$	95	$74.9_{\blacktriangledown 3.5}$	85	$47.5_{\blacktriangledown 1.2}$
w/o retirement	132	$91.4_{\blacktriangledown 1.0}$	121	$75.9_{\blacktriangledown 2.5}$	111	$47.9_{\blacktriangledown 0.8}$
w/o mutation	44	$91.8_{\blacktriangledown 0.6}$	66	$76.8_{\blacktriangledown 1.6}$	41	$48.2_{\blacktriangledown 0.5}$

5.2 Main Results

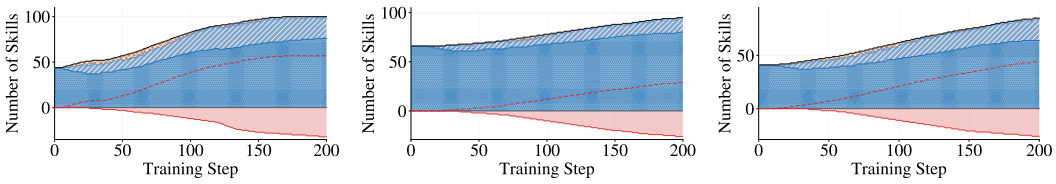
As shown in Table 2, SKILLFORGE achieves 92.4% on ALFWorld and 78.4% on WebShop, outperforming the closed-source GPT-4o and Gemini-2.5-Pro baselines by a wide margin and improving over SkillRL by 2.8% relative on ALFWorld and 7.8% relative on WebShop. The largest per-subtask relative improvement over SkillRL is on ALFWorld's Look at 21.3%; the Clean subtask remains within 0.2% of SkillRL, within sampling noise. On Search-Augmented QA, as illustrated in Figure 2a, SKILLFORGE reaches $48.7 \pm 0.3\%$ micro-averaged accuracy on the full test set, the highest overall among all methods; per-dataset results are in Appendix D.

As illustrated in Figure 2b, SKILLFORGE and SkillRL both climb from ∼60% to ∼70% between steps 50 and 65, but SKILLFORGE separates beyond step 75 as the forging lifecycle takes effect, reaching ∼90% while SkillRL oscillates in the ∼80–88% range over the rest of the training budget.

5.3 Ablation Study

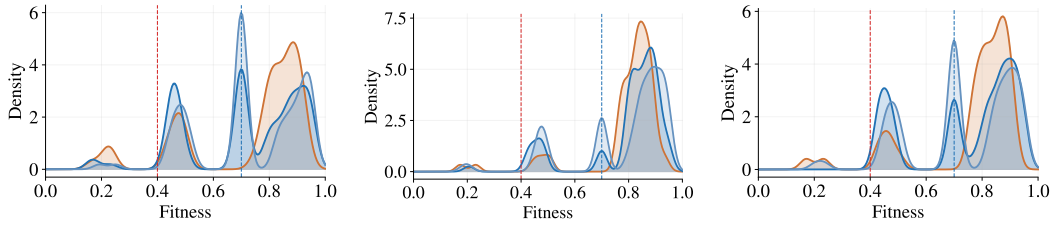
As shown in Table 3, removing the entire SKILLFORGE lifecycle produces the largest relative drops on every environment, peaking at 7.4% on WebShop. Pre-retirement is the single most impactful component, with per-stage relative drops of 1.8%, 4.5%, and 2.5% on ALFWorld, WebShop, and Search-Augmented QA respectively. The single-component drops consistently sum to more than the joint w/o forging drop, consistent with sub-additive interactions among the three components.

Larger libraries do not imply better performance. Without retirement the library grows to over 130 skills on ALFWorld, yet SKILLFORGE outperforms it with a compact set of 100. On WebShop,



(a) ALFWorld (b) WebShop (c) Search-Augmented QA

Figure 3: **Skill library dynamics** across three environments. Stable, Active, Trial: lifecycle states (stacked areas). Retired: cumulative retired (below zero). —: total non-retired; - -: mutated (gen≥1).



(a) ALFWorld (b) WebShop (c) Search-Augmented QA

Figure 4: **Fitness distribution** at three training stages. Step 40, Step 120, Step 195: KDE at each stage. - -: $\delta_{\text{retire}} = 0.4$; - -: $\delta_{\text{stable}} = 0.7$. Retirement depletes the left tail, mutation populates the middle band via newly born skills, and stabilization maintains the high-fitness regime.

enabling pre-retirement and mutation lifts the static 72.6% baseline to 75.9%; adding retirement trims the library from 121 skills to 95 and closes the remaining 3.2% relative gap to the full 78.4%, confirming that quality control, not scale, drives the gain.

5.4 Forging Dynamics

As shown in Figure 3, the skill library evolves similarly across the three environments. On ALFWorld, 200 training steps grow the library to 100 non-retired skills, with 132 created and 32 retired; the cap binds near step 170, after which mutations and retirements balance into dynamic equilibrium. At the final checkpoint 76 skills are `stable` and 24 `active` or `trial`.

As illustrated in Figure 4, the distribution evolves through two opposing flows rather than a monotone rightward shift. Retirement depletes the left tail: on ALFWorld the sub-δ_{retire} fraction shrinks from 7.5% to 2.0% across training, while mutation expands the middle band $[\delta_{\text{retire}}, \delta_{\text{stable}})$ from 18.9% to 22.0% as children enter with a neutral prior at 0.5. The stable regime retains most of the library throughout and mean fitness stays near 0.73, an equilibrium in which pruning and churn rebalance. Appendix E provides concrete mutation examples and the deepest forging lineages.

6 Conclusion

SKILLFORGE treats skill-library management as a fitness-driven forging process in which skills and the model co-evolve under online RL through retirement, stabilization, and LLM-guided mutation, delivering up to 7.8% relative improvement over the strongest memory-augmented RL baseline across ALFWorld, WebShop, and Search-Augmented QA while keeping the active library compact. We construct SKILLFURNACE, a 5k+ annotated record bundle of retirement-filtered SFT trajectories, evolved skill libraries with fitness annotations, and retirement events with human-annotated failure categories, in which procedural rigidity emerges as the dominant mode by which once-useful skills turn into active interference, the empirical signature of *delayed obsolescence*.

7 Limitations

Each configuration is trained with a single RL seed; the reported $\pm$ values aggregate five evaluation sampling seeds rather than independent training runs. Fitness is assigned at the episode level, so the K co-retrieved skills share one binary reward; per-skill leave-one-out attribution agrees with this estimate on 93% of retirement decisions, but the approximation may slow convergence where co-retrieval is structurally unavoidable. All experiments use a 7B base model, so lifecycle dynamics at larger scales remain open. Mutation relies on an external teacher; self-mutation by the base model still outperforms SkillRL but forgoes the mutation gain. SKILLFURNACE is drawn from three environments. More broadly, delayed obsolescence can affect any context accumulated for an improving agent, such as retrieved passages or tool catalogs, which a fitness-driven lifecycle could manage in the same way.

Acknowledgments and Disclosure of Funding

This work is supported in part by the National Key R&D Program of China under Grant No. 2025YFC3309703, the Beijing Natural Science Foundation No. 4262033, and the National Natural Science Foundation of China under Grant Nos. U25B2076, 62441229, and 62377043. The authors declare no competing interests.

References

[1] Anthropic. Claude Code. https://www.anthropic.com/claude-code, 2025.

[2] OpenClaw. OpenClaw. https://github.com/openclaw/openclaw, 2026.

[3] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[4] Jiaqi Liu, Yaofeng Su, Peng Xia, Yiyang Zhou, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. SimpleMem: Efficient lifelong memory for LLM agents. *arXiv preprint arXiv:2601.02553*, 2026. URL https://github.com/aiming-lab/SimpleMem.

[5] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *ArXiv*, abs/2508.06433, 2025.

[6] Yongjin Yang, Sinjae Kang, Juyong Lee, Dongjun Lee, Se young Yun, and Kimin Lee. Automated skill discovery for language agents through exploration and iterative feedback. *ArXiv*, abs/2506.04287, 2025.

[7] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

[8] Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

[9] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

[10] Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, et al. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory. *ArXiv*, abs/2601.03192, 2026.

[11] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[12] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651, 2023.

[13] Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

[14] 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, pages 19632–19642, 2024.

[15] 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. *arXiv preprint arXiv:2305.16291*, 2023.

[16] Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[17] Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran L. Li. Proposer-Agent-Evaluator (PAE): Autonomous skill discovery for foundation model internet agents. *ArXiv*, abs/2412.13194, 2024.

[18] Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: Advanced reasoning and learning for autonomous AI agents. *arXiv preprint arXiv:2408.07199*, 2024.

[19] Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

[20] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

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

[22] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. EvoPrompt: Connecting LLMs with evolutionary algorithms yields powerful prompt optimizers, 2025. URL https://arxiv.org/abs/2309.08532.

[23] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

[24] Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through AI feedback. *arXiv preprint arXiv:2310.13032*, 2023.

[25] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468 – 475, 2023.

[26] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[27] OpenAI. Introducing o3 and o4-mini, 2025. URL https://openai.com/index/introducing-o3-and-o4-mini/.

[28] Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan-ang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

[29] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

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

[31] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[32] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural Questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

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

[34] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 9802–9822, 2023.

[35] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

[36] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

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

[38] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

[39] 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 *The eleventh international conference on learning representations*, 2023.

[40] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers), pages 12248–12267, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

[41] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438, 2025.

[42] DeepSeek Team. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *ArXiv*, abs/2501.12948, 2025.

[43] Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. ZeroSearch: Incentivize the search capability of LLMs without searching. *arXiv preprint arXiv:2505.04588*, 2025.

[44] Xuhui Zheng, Kang An, Ziliang Wang, Yuhang Wang, and Yichao Wu. StepSearch: Igniting LLMs search ability via step-wise proximal policy optimization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21805–21830, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1106. URL https://aclanthology.org/2025.emnlp-main.1106/.

[45] OpenAI. GPT-4o system card, 2024. URL https://openai.com/index/gpt-4o-system-card/.

[46] Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *ArXiv*, abs/2507.06261, 2025.

[47] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pages 1279–1297. ACM, March 2025. doi: 10.1145/3689031.3696075.

[48] Qwen Team. Qwen technical report. *ArXiv*, abs/2309.16609, 2023.

[49] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, et al. Kimi K2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

[50] Z.ai. GLM-5.2: Built for long-horizon tasks, 2026. URL https://z.ai/blog/glm-5.2.

[51] MiniMax. MiniMax M3: Frontier coding, 1M context, native multimodality, 2026. URL https://www.minimax.io/blog/minimax-m3.

A Notation

Table 4 summarizes the symbols used throughout the paper, grouped by domain.

Table 4: Notation used throughout the paper.

Symbol	Definition	Description
Environment and Agent		
d	Task description	Query input to the agent; d_τ denotes the instance tied to trajectory τ
$\mathcal{D}_{\text{task}}$	Training task distribution	Uniform over per-environment training tasks
R	Reward function	$R(\tau) \in \{0, 1\}$, binary task success
π_θ	Agent policy	Parameterized by θ
π_{θ_0}	Base model policy	Before any fine-tuning
$\pi_{\theta_{\text{old}}}$	Old policy (GRPO)	Generates rollouts; updated each training step
$\pi_{\theta_{\text{sft}}}$	SFT checkpoint policy	RL initialization and reference policy π_{ref}
$\tau = (p_1, a_1, o_2, \ldots)$	Episode trajectory	Contexts, actions, observations over one episode
a_t	Agent action at step t	Sampled as $a_t \sim \pi_\theta(\cdot \mid p_t)$
o_t	Observation at step t	Current environment observation
p_t	Agent context at step t	Built via Eq. (2); $p^{(j)}$ uses library $\mathcal{S}^{(j)}$ at training step j
h_t	Interaction history	Sliding window of the last W=5 (o, a) pairs
Skill Library		
$s_i = (v, b, \omega)$	Skill	Name, behavioral principle, applicability condition
$\mathcal{S}$	Skill library	Set of all non-retired skills
$\mathcal{S}_0$	Initial skill library	Input to Eq. (3); $\mathcal{S}^{(0)}_{\text{retire}}$ is the pre-retired subset
$\mathcal{S}^{(k)}$	Library at step k	After k forging updates; $\mathcal{S}^{(0)} = \mathcal{S}_0 \setminus \mathcal{S}^{(0)}_{\text{retire}}$
K	Retrieval count	Number of skills retrieved per episode
$\phi(d, \mathcal{S})$	Retrieval function	Selects top-K skills for task d; ϕ_{eval} is the pre-retirement variant
$\mathcal{S}_\tau$	Retrieved skill set	$\phi(d, \mathcal{S})$ for trajectory τ
U_i	Usage count	Cumulative retrievals of s_i; superscript (0) marks pre-retirement value
C_i	Success count	Cumulative episodes in which s_i was retrieved and succeeded
f_i	Skill fitness	C_i/U_i when $U_i \geq N_{\text{warm}}$, else f_{default}
f_{default}	Default fitness	Neutral prior for skills with $U_i < N_{\text{warm}}$
$\hat{f}_i$	Proto-fitness	$C_i^{(0)}/U_i^{(0)}$ from pre-retirement rollouts; equals $f_i^{(0)}$
$f_i^{(k)}$	Fitness at step k	Non-stationary estimate (Eq. (11))
$\ell(s_i)$	Lifecycle state	$\in \{\texttt{trial}, \texttt{active}, \texttt{stable}, \texttt{retired}\}$
g_i	Generation	Mutation depth; $g_i = 0$ for distilled skills
$\mathcal{M}_{\text{cand}}$	Mutation candidate pool	Active skills with moderate fitness (Eq. (10))
$\mathcal{M}_T$	Teacher model	Generates mutations via $\mathcal{P}_\mu$
$\mathcal{P}_\mu$	Mutation template	Assembles parent skill, fitness, and failures
$\mathcal{T}_{\text{fail}}$	Failure trajectories	Recent failed episodes used as mutation context
$\mathcal{D}_{\text{SFT}}$	SFT dataset	Successful trajectories filtered by retirement (Eq. (4))
$\text{Forge}(\cdot)$	Forging operator	Full forging cycle (Algorithm 1)
Hyperparameters		
M	Preliminary eval episodes	Rollout count for pre-retirement
δ_{pre}	Pre-retirement threshold	Proto-fitness below which initial skills are discarded
δ_{retire}	Retirement threshold	Fitness boundary below which active skills are retired
δ_{stable}	Stabilization threshold	Fitness boundary for active $\rightarrow$ stable
δ_{demote}	Demotion threshold	Fitness boundary for stable $\rightarrow$ active
$[\delta_{\text{low}}, \delta_{\text{high}}]$	Mutation fitness range	Bounds for mutation eligibility
N_{min}	Proto-fitness min usage	Min usage before proto-fitness is evaluated
N_{warm}	Fitness warm-up	Min usage before fitness replaces f_{default}
N_{promote}	Promotion min usage	Min usage for trial $\rightarrow$ active
N_{stable}	Stabilization min usage	Min usage for active $\rightarrow$ stable
N_{mutate}	Mutation min usage	Min usage for mutation eligibility
N_{retire}	Retirement min usage	Min usage before an active skill can be retired
$N_{\text{retire}}^{(g_i)}$	Generation-aware retirement	N_{protect} if $g_i = 0$, else N_{retire}
N_{protect}	Protection threshold	Higher usage bar for generation-0 skills
K_{retire}	Retirement budget	Max retirements per forging cycle
K_{mutate}	Mutation budget	Max mutations per forging cycle
S_{max}	Skill cap	Maximum library size
E	Forging period	Validation epochs between forging cycles
W	History window size	Recent (o, a) pairs kept in h_t; W=5
GRPO Training		
$\mathcal{L}_{\text{SFT}}$	SFT loss	Cross-entropy on retirement-filtered trajectories (Eq. (5))
G	Group size	Trajectories sampled per task
$\hat{A}_n$	Group-normalized advantage	$(r_n - \mu_G)/\sigma_G$
ϵ	Clipping range	PPO-style surrogate clipping
β	KL coefficient	Regularization against reference policy
τ_n	Episode n in a GRPO group	Used for $\hat{A}_n$ and reward $r_n = R(\tau_n)$
$\boldsymbol{y}_n$	Token sequence of τ_n	Action tokens produced autoregressively by π_θ
$\tau^{(j)}$	Trajectory at step j	Rollout from π_{θ_j} for the non-stationary fitness (Eq. (11))
$y_{n,l}$	Token at position l	l-th action token in $\boldsymbol{y}_n$
$c_{n,l}$	Left context of $y_{n,l}$	Environment-step context plus earlier tokens
$\rho_{n,l}$	Importance ratio	$\pi_\theta(y_{n,l} \mid c_{n,l})/\pi_{\theta_{\text{old}}}(y_{n,l} \mid c_{n,l})$

B Implementation Details

Training configuration. Table 5 lists the training hyperparameters and hardware setup. The base model is Qwen2.5-7B-Instruct [48], fine-tuned with GRPO on 64× NVIDIA H200 GPUs using the verl framework. The initial skill library is environment-specific: ALFWorld uses 44 skills (12 general, 21 task-specific, 11 common-mistake), WebShop uses 66 skills (15 general, 39 task-specific, 12 common-mistake), and Search-Augmented QA uses 41 skills (10 general, 20 query-type, 11 common-mistake), all inherited from the SkillRL release [8]. General skills encode cross-task heuristics (e.g., "always verify inventory after pickup"), task-specific skills target individual task types, and common-mistake skills capture frequently observed failure patterns from initial rollouts (e.g., "do not leave an object behind after finding it").

Evaluation sample sizes. The three environments follow the setup of SkillRL [8], our most direct baseline, and cover three distinct task types: embodied control, web navigation, and retrieval-based QA. ALFWorld reports per-sub-task accuracy on the 130-task unseen-test set, partitioned as Pick (24), Look (15), Clean (27), Heat (26), Cool (21), and Pick2 (17); the **All** column in Table 2 reports overall success on this set, and baseline rows are taken from SkillRL [8]. WebShop uses the standard 500-task held-out test set for both Score and Succ. Search-Augmented QA per-sub-dataset sample sizes appear in Table 10.

Table 5: Training configuration for SKILLFORGE. All three environments share the GRPO optimizer, group size, and training budget; hardware is identical across runs.

Hyperparameter	**Value**
Base model	Qwen2.5-7B-Instruct
GRPO learning rate	1×10^{-6}
Batch size	16
Group size G	8
Clipping range ϵ	0.2
KL coefficient β	0.001
Training steps	200
History window size W	5
Sampling temperature (train & eval)	1.0
Validation epoch	5 steps
Hardware	64× NVIDIA H200
Framework	verl

Forging hyperparameters. Table 6 lists all forging-specific hyperparameters. All thresholds and budgets are shared across the three environments rather than tuned per task; only M and $S_{\max}$ scale with the size of the seed library. The pre-retirement threshold $\delta_{\text{pre}} = 0.3$ is looser than the runtime $\delta_{\text{retire}} = 0.4$ because the pre-retirement phase uses the unadapted base policy π_{θ_0}, whose success rates are systematically lower than those observed during RL; a lower bar avoids discarding borderline-but-potentially-useful skills before RL has had a chance to re-evaluate them. Two usage-count thresholds also deserve clarification. The proto-fitness minimum $N_{\min} = 3$ (Eq. (3)) is smaller than the runtime warm-up $N_{\text{warm}} = 5$ (Eq. (6)) because proto-fitness is estimated under the fixed base policy over M dedicated rollouts, so three observations already yield a low-variance estimate for the binary filter $\hat{f}_i < \delta_{\text{pre}}$. Runtime fitness, in contrast, is estimated under a moving policy and feeds into continuous lifecycle transitions (promotion, stabilization, demotion, retirement), which justifies a slightly more conservative warm-up. Correspondingly, Eq. (3) is a hard *filter*: skills without $U_i^{(0)} \geq N_{\min}$ are simply not candidates for removal and remain in $\mathcal{S}^{(0)}$. In contrast, Eq. (6) must emit a numeric score for retrieval and ranking, so it falls back to the neutral prior $f_{\text{default}} = 0.5$ when the usage count is too low to trust the empirical rate. A second boundary case arises because $\delta_{\text{low}} = \delta_{\text{retire}} = 0.4$: a skill with f_i exactly equal to this value satisfies both the retirement condition (strict inequality $f_i < \delta_{\text{retire}}$ is false, so retirement does not fire) and the mutation range (closed interval $[\delta_{\text{low}}, \delta_{\text{high}}]$ includes the endpoint), so such a skill is eligible for mutation but not retirement and behaves identically to skills with f_i slightly above 0.4.

Table 6: Forging hyperparameters for SKILLFORGE. Slash-separated values apply to ALFWorld, WebShop, and Search-Augmented QA respectively; S_{max} and M scale with the initial library size of each environment.

Hyperparameter	Symbol	Value
Proto-fitness min usage	N_{min}	3
Pre-retirement threshold	δ_{pre}	0.3
Preliminary eval episodes	M	500 / 450 / 400
Fitness warm-up threshold	N_{warm}	5
Default fitness (new skills)	f_{default}	0.5
Promotion min usage	N_{promote}	10
Retirement fitness threshold	δ_{retire}	0.4
Retirement min usage	N_{retire}	20
Stabilization fitness threshold	δ_{stable}	0.7
Stabilization min usage	N_{stable}	30
Demotion fitness threshold	δ_{demote}	0.5
Mutation fitness range	$[\delta_{\text{low}}, \delta_{\text{high}}]$	$[0.4, 0.7]$
Mutation min usage	N_{mutate}	5
Max mutations per cycle	K_{mutate}	5
Max retirements per cycle	K_{retire}	3
Total skill cap	S_{max}	100 / 95 / 85
Original skill protection	N_{protect}	50
Forging frequency	E	2 (every 10 training steps)
Teacher model	$\mathcal{M}_T$	Kimi-K2.5 [49]
Skill retrieval	top-K	6 (task-type matching)

Lifecycle well-definedness. The four branches of Eq. (9) have pairwise disjoint guards: they require $\ell(s_i) \in \{\texttt{trial}, \texttt{active}, \texttt{stable}, \texttt{active}\}$ respectively, and the two `active` branches trigger on disjoint fitness ranges, one requiring $f_i \geq \delta_{\text{stable}}$ and the other $f_i < \delta_{\text{retire}}$, so at most one branch fires per skill per cycle. Eq. (9) is declarative; Algorithm 1 specifies the per-cycle ordering, with promotion and demotion preceding retirement and stabilization preceding mutation.

Mutation weighting. We use the simplest monotone-decreasing weighting, $P(s_i) \propto 1 - f_i$, rather than sharper alternatives such as $1/f_i$ or $\exp(-\lambda f_i)$. Because mutation candidates already satisfy $\delta_{\text{low}} \leq f_i \leq \delta_{\text{high}}$ (default $[0.4, 0.7]$), the weights $1 - f_i$ lie in $[0.3, 0.6]$, giving at most a $2\times$ preference for lower-fitness parents. This is enough to bias selection toward skills with more room to improve without concentrating mutation on a single outlier. Empirically, both $1/f_i$ and uniform sampling lower average success (Table 14).

Threshold sensitivity. Table 7 sweeps the two most load-bearing thresholds, the retirement threshold δ_{retire} and the library cap S_{max}, around their default values on ALFWorld. The overall success rate varies within $\pm 1.4\%$ across the swept range, showing robustness to threshold choices. We observe a mild concavity: overly aggressive retirement ($\delta_{\text{retire}} = 0.5$) removes borderline-but-useful skills before they stabilize, while overly permissive retirement ($\delta_{\text{retire}} = 0.3$) retains low-fitness skills that dilute the retrieved context. The default $\delta_{\text{retire}} = 0.4$ sits near the peak of this trade-off.

Table 7: Sensitivity of overall ALFWorld success (%) to the two most load-bearing thresholds, sweeping one threshold at a time around its default. Default values are boxed.

Retirement threshold δ_{retire} (default 0.4)				
δ_{retire}	0.3	$\boxed{0.4}$	0.5	0.6
Success (%)	91.3	**92.4**	91.8	91.0
Library cap S_{max} (default 100)				
S_{max}	70	85	$\boxed{100}$	130
Success (%)	91.0	91.9	**92.4**	91.6

Compute resources. All RL runs use 64× NVIDIA H200 GPUs under the verl framework. Estimated wall-clock per 200-step training run is ≈24 hours for ALFWorld, ≈18 hours for WebShop, and ≈72 hours for Search-Augmented QA (retrieval-dominated), for a total of ≈114 wall-clock hours × 64 GPUs = 7,296 GPU-hours across the three environments per full training run. The pre-retirement evaluation phase (Section 4.1) adds ≈6 hours per environment on the same hardware. Evaluation is run five times per reported number (sampling seeds $\{30, 40, 42, 50, 62\}$) and adds ≈1.5–4.5 hours per environment per seed. Teacher model overhead is negligible relative to RL: a forging cycle runs every $E = 2$ validation epochs (every 10 training steps) and issues up to $K_{\text{mutate}} = 5$ mutation calls per cycle, for at most $\lceil 200/10 \rceil \times 5 = 100$ mutation calls per training run, all dispatched asynchronously during validation so they do not block GPU training. The teacher $\mathcal{M}_T$ is treated as a frozen black-box LLM: any sufficiently capable instruction-tuned model (proprietary or open-weight) can serve in this role, since the interface is only a structured template $\mathcal{P}_\mu$ and a single-shot text response. We use Kimi-K2.5 here for mutation quality, but the framework imposes no dependence on a specific teacher: GLM-5.2 [50] and MiniMax M3 [51] perform on par with it (Table 12).

Mutation template. The mutation operator (Section 4.3, Eq. (10)) assembles the parent skill's metadata, fitness statistics, and recent failure examples into the following structured template $\mathcal{P}_\mu$, which is sent to the teacher model $\mathcal{M}_T$:

Mutation Template $\mathcal{P}_\mu$

```
You are improving a skill for an embodied AI agent in {environment}.

The following skill has moderate performance (fitness={f_i}, used {U_i} times,
succeeded {C_i} times):

Original skill:
- ID: {skill_id}
- Title: {title}
- Principle: {principle}
- When to apply: {when_to_apply}

[If recent failures exist:]
This skill was involved in the following failed episodes:
- Task: {task description}
  Agent action: {failed action}
- ...
Please analyze these failure patterns and generate an improved skill that
specifically addresses them.

Generate ONE improved variant that addresses the weakness shown by its
moderate success rate. The improved skill should be more specific,
actionable, or correct than the original.

Return ONLY a JSON object with these fields:
{"skill_id": "{new_id}", "title": "...", "principle": "...",
"when_to_apply": "..."}
```

The template includes up to 8 recent failure examples (task description and agent action, each truncated to 200 characters) from episodes where the parent skill was retrieved but the episode failed. Failure examples are omitted when no failures are recorded. The teacher model returns a single JSON object, which is parsed and assigned a new skill identifier.

C SKILLFURNACE Dataset Details

SKILLFURNACE is a three-part dataset constructed alongside SKILLFORGE. Each part draws from a distinct data source within our experimental pipeline: (i) **SFT trajectories** come from rollouts of the Qwen2.5-7B-Instruct base model during the preliminary evaluation phase, before any RL training has begun; (ii) **evolved skill libraries** are initialized from the seed library released by SkillRL [8] and then logged at every forging cycle as one record per skill per snapshot; (iii) **retirement events** are lifecycle-gate decisions aggregated across the primary training run and additional previously recorded training runs (ablation and sensitivity variants), so that the event count is not limited to a single run. Every mined event is labelled and quality-controlled under the same criteria as the primary

run. In total, SKILLFURNACE contains **5,852 annotated records** across the three environments: 1,013 SFT trajectories, 4,521 library snapshot records, and 318 retirement events, summarized in Table 1. Table 8 summarizes the library component across environments, Table 1 reports dataset scale and retirement-event statistics, and Table 9 breaks retirement events down by failure category per environment.

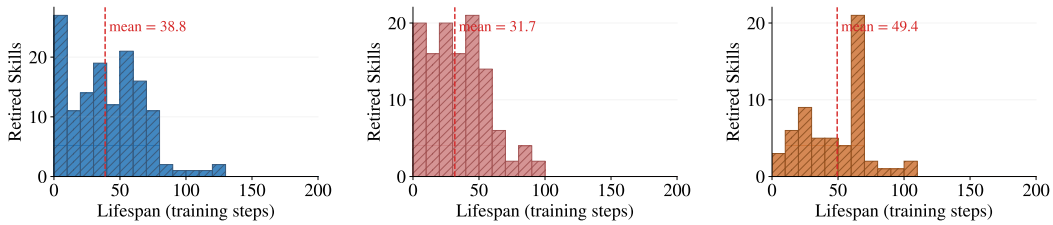
Table 8: SKILLFURNACE evolved-library composition per environment. Initial $|\mathcal{S}_0|$ is the seed library released by SkillRL [8]; *total ever created* counts every skill ever introduced, including all mutation children; *retired during training* counts skills that reached `retired`; *final non-retired* is the active library size at the last training step. The final row shows what fraction of the surviving library originated from LLM-guided mutation.

Statistic	ALFWorld	WebShop	Search-Aug. QA	Total
Initial $\lvert\mathcal{S}_0\rvert$	44	66	41	151
Total ever created	132	121	111	364
Retired during training	32	26	26	84
Final non-retired	100	95	85	280
of which `stable`	76	80	64	220
of which mutated ($g_i \geq 1$)	57	29	44	130

SFT trajectories. The preliminary evaluation phase executes M rollout episodes with the Qwen2.5-7B-Instruct base model π_{θ_0} and retrieved skills injected via Eq. (2). Per-environment M values are 500 (ALFWorld), 450 (WebShop), and 400 (Search-Augmented QA); see Table 6 for all forging-related hyperparameters. We retain every successful trajectory whose retrieved-skill set is disjoint from the pre-retired set $\mathcal{S}^{(0)}_{\text{retire}}$, yielding the SFT corpus $\mathcal{D}_{\text{SFT}}$ defined in Eq. (4): 412 trajectories in ALFWorld, 348 in WebShop, and 253 in Search-Augmented QA, for a total of 1,013 SFT trajectories. Each trajectory is stored as an ordered sequence of context–action pairs with the retrieved-skill ids attached to each context, so the base corpus can be reconstructed and re-filtered under alternative retention rules without rerunning rollouts. All trajectories in this component come exclusively from the base Qwen2.5 model, making them a clean cold-start fine-tuning set decoupled from any specific RL setup.

Evolved skill libraries. Every forging cycle emits a library snapshot recording, for each skill, the id, name, behavioral principle b, applicability condition ω, cumulative usage count U_i, cumulative success count C_i, current fitness f_i, lifecycle state $\ell(s_i) \in \{\texttt{trial}, \texttt{active}, \texttt{stable}, \texttt{retired}\}$, generation depth g_i, and parent id for mutated skills. The initial library $\mathcal{S}_0$ logged at snapshot 0 is the seed library released by SkillRL [8]; subsequent snapshots track how this seed evolves under RL. Forging runs every 10 training steps over the 200-step budget, producing 21 snapshots per run. Each snapshot emits one record per skill tracked at that step, yielding 1,512 records in ALFWorld, 1,683 in WebShop, and 1,326 in Search-Augmented QA, for a total of 4,521 library snapshot records. Table 8 reports the composition at the last training step. Across the three environments, the library grows from 44, 66, and 41 seed skills to a final non-retired size of 100, 95, and 85 at the cap S_{max}, with 132, 121, and 111 skills created cumulatively. Of the final non-retired skills, 76, 80, and 64 reach the `stable` state, and 57, 29, and 44 originate from LLM-guided mutation ($g_i \geq 1$), making 46.4% of final skills across the three environments children rather than seed skills. Snapshots are stored together with per-cycle forging logs, so any fitness trajectory or lifecycle transition can be reconstructed offline.

Retirement events. Each retirement event, whether triggered by the retirement gate or by the library cap S_{max}, records the full autopsy of a removed skill: the skill content, the complete fitness history leading up to retirement, the lifecycle-transition log, parent and sibling skills from the mutation tree, a human-annotated failure category drawn from the 8-way taxonomy described in Appendix E.2, and a free-text *key flaw* field describing the retired skill's failure mode. To make the sample size independent of any single training configuration, events are aggregated from the primary training run and additional previously recorded training runs; the per-environment totals in Table 1 therefore exceed the primary-run retirement counts reported in Section 5.4 by design. Retirement events exhibit an average peak fitness of 0.67 and an average lifespan of 38.0 training steps, with fitness dropping by 0.28 on average at retirement; 57.2% first reached the `stable` state before this decay, the defining pattern of *delayed obsolescence* that motivates the forging lifecycle. Figure 5 shows



(a) ALFWorld (b) WebShop (c) Search-Augmented QA

Figure 5: **Lifespan distribution of retirement events in SKILLFURNACE**, measured as `retired_step` − `born_step`. Retirements spread across the full training horizon rather than clustering at any single stage; the red dashed line marks the per-environment mean.

the lifespan distribution: retirements spread across the full training horizon rather than clustering at any single stage, so the retirement gate fires throughout RL rather than only at initialization or endgame. Table 9 reports the per-environment breakdown across the eight categories. Overly rigid ordering is the dominant mode in every environment (74.6% / 77.7% / 67.8%), confirming that procedural rigidity rather than content omission is the principal failure mode of LLM-distilled skills. Environment-specific patterns also emerge: *hallucinated action* accounts for 10.7% of WebShop events, corresponding to skills that reference UI elements absent from the WebShop environment, and *contradicts environment* is exclusive to Search-Augmented QA, corresponding to skills that invoke search operations the environment does not expose.

Table 9: SKILLFURNACE retirement events by failure category (8-way human-annotated taxonomy). Percentages are within environment. *Overly rigid ordering* dominates in every environment, confirming that procedural rigidity, not content omission, is the principal failure mode of LLM-distilled skills. Environment-specific patterns also emerge: *hallucinated action* appears only in WebShop (UI elements absent from the environment) and *contradicts environment* appears only in Search-Augmented QA (search operations absent from the environment).

Failure category	**ALFWorld**	**WebShop**	**Search-Aug. QA**	**Total**
Overly rigid ordering	103 (74.6%)	94 (77.7%)	40 (67.8%)	237 (74.5%)
Overly specific	26 (18.8%)	2 (1.7%)	2 (3.4%)	30 (9.4%)
Premature specialization	6 (4.3%)	10 (8.3%)	0 (0.0%)	16 (5.0%)
Hallucinated action	0 (0.0%)	13 (10.7%)	0 (0.0%)	13 (4.1%)
Redundant with policy	2 (1.4%)	1 (0.8%)	7 (11.9%)	10 (3.1%)
Contradicts environment	0 (0.0%)	0 (0.0%)	7 (11.9%)	7 (2.2%)
Scope mismatch	0 (0.0%)	0 (0.0%)	3 (5.1%)	3 (0.9%)
Vague or tautological	1 (0.7%)	1 (0.8%)	0 (0.0%)	2 (0.6%)
Total	**138**	**121**	**59**	**318**

Data format. SKILLFURNACE is organized as three top-level JSON files with a schema document: `sft_trajectories.json`, `library_snapshots.json`, and `retirement_events.json`. All records are keyed by a global `skill_id` so the three components can be joined: every skill id appearing in `retirement_events.json` is also present in the matching run's `library_snapshots.json` for every step between birth and retirement, and every skill retrieved in `sft_trajectories.json` is resolvable in the step-0 library snapshot.

D Search-Augmented QA Full Per-Dataset Results

Table 10 reports accuracy on each of the seven Search-Augmented QA sub-datasets that are summarised by the radar chart in Figure 2a. Both in-domain datasets (†: NQ, HotpotQA) and out-of-domain datasets (⋆: TriviaQA, PopQA, 2WikiMultiHopQA, MuSiQue, Bamboogle) are included. The 48.7% aggregate number reported in the main text corresponds to micro-averaged accuracy over the 51,713-sample concatenation of these seven test sets.

Table 10: Per-dataset accuracy on Search-Augmented QA (%). †: in-domain; ⋆: out-of-domain. Per-dataset cells for non-SKILLFORGE rows are taken from their respective source papers. The **Overall** column is recomputed as the sample-weighted micro-average over the 51,713-sample union shown in *Test samples*, so that all methods are compared on an identical test set; these values may differ from source-paper totals that used different evaluation configurations. **Bold**: best; underline: second best. "–": not reported.

Method *Test samples*	**NQ**† *3,610*	**TriviaQA**⋆ *11,313*	**PopQA**⋆ *14,267*	**HotpotQA**† *7,405*	**2Wiki**⋆ *12,576*	**MuSiQue**⋆ *2,417*	**Bamboogle**⋆ *125*	**Overall** *51,713*
In-context / retrieval baselines								
Vanilla [48]	11.6	35.6	1.2	16.4	22.2	4.8	14.4	16.9
CoT	12.8	35.6	3.8	16.2	22.6	6.6	24.0	17.9
RAG	27.4	58.2	17.8	25.8	23.2	9.4	16.8	29.4
Search-o1 [41]	19.4	40.6	11.4	17.0	27.0	8.6	30.4	22.9
R1-Instruct [42]	21.0	44.9	17.1	20.8	27.5	6.0	19.2	26.0
RL-based search methods								
Search-R1 [31]	39.3	61.0	39.7	37.0	40.1	14.6	36.8	42.9
ZeroSearch [43]	43.6	61.8	**51.5**	34.6	35.2	18.4	27.8	45.2
StepSearch [44]	–	–	–	38.6	36.6	**22.6**	40.0	–
EvolveR [9]	43.5	63.4	44.6	38.2	**42.0**	15.6	54.4	45.8
SkillRL [8]	45.9	63.3	45.9	43.2	40.3	20.2	73.8	46.8
SKILLFORGE (Ours)	**48.2**	**65.0**	50.1	**43.9**	40.4	20.3	**77.2**	**48.7**

SKILLFORGE obtains the best or second-best accuracy on every sub-dataset, with the largest absolute improvements on the multi-hop benchmarks Bamboogle (+3.4 over the strongest baseline) and HotpotQA (+0.7). Gains are consistent on both in-domain and out-of-domain splits, indicating that the forging lifecycle does not overfit to the training distribution.

E Case Studies

E.1 Deepest Forging Lineages

As shown in Figure 6, mutation lineages exhibit a heavy-tailed depth distribution: most refinements converge to a stable variant within one or two generations, while a tail extends up to nine generations when niches remain. Useful variants stabilize and stop mutating, so only lineages with room to specialize keep branching.

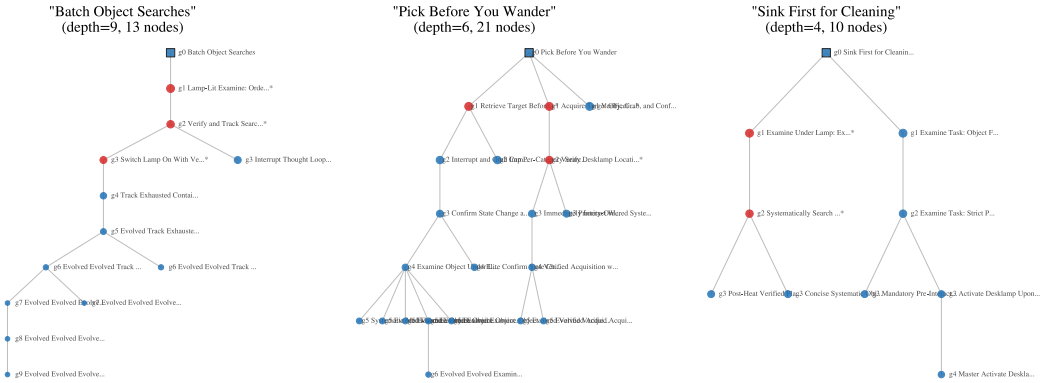


Figure 6: **Deepest forging lineages** on ALFWorld. Each subplot shows a root skill (gen=0, square) and its multi-generation descendants (circles). Red = retired, blue = stable. Early variants are often retired while later, more refined variants survive.

E.2 Failure Taxonomy Details

As shown in Figure 7, the 318 retirement events were labelled into one of eight failure categories. Two annotators labelled the events under this 8-way taxonomy; an initial pass reached Cohen's $\kappa = 0.5$ inter-annotator agreement, and remaining disagreements were resolved by discussion. The

dominant failure mode is *overly rigid ordering* at 74.5%, where the skill imposes a fixed procedural sequence that fails for edge cases; the second most common is *overly specific* at 9.4%, where skills work for narrow task subsets but fail broadly. Only 10 of the 318 retirement events (3.1%) are labeled *redundant with policy*; the remaining 96.9% contain prescriptions inconsistent with successful rollouts—rigid ordering, hallucinated operations, overfit specializations, self-contradictions—so *delayed obsolescence* reflects once-stable skills becoming concretely misleading, not the policy outgrowing harmless crutches. Environment-specific patterns also emerge: *contradicts environment* failures appear exclusively in Search-Augmented QA, where LLM-guided mutation hallucinates search operations absent from the environment.

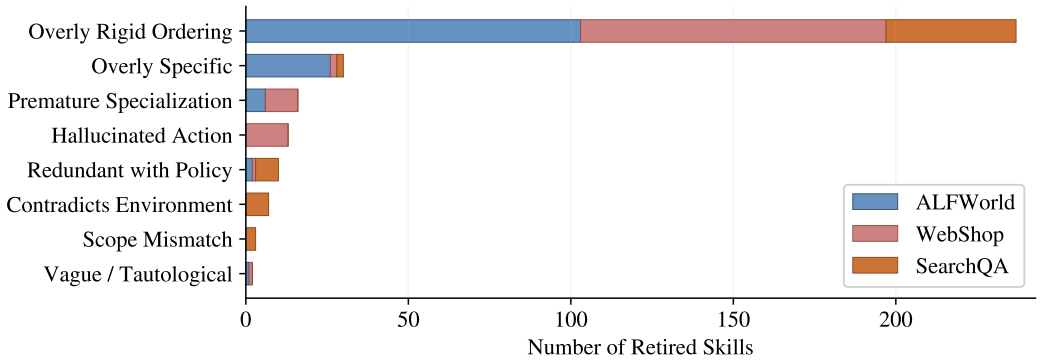


Figure 7: Failure taxonomy of 318 retirement events across three environments. Blue: ALFWorld; red: WebShop; orange: Search-Augmented QA. Overly rigid ordering dominates across all environments; environment-specific failure modes emerge in Search-Augmented QA.

E.3 WebShop: Hidden-Constraint Verification

> **Parent skill** ("Confirm Hidden Specs", gen=0):
> *Principle*: Scroll to bullet points/description to verify non-title attributes (material, care, fit, design features); reject items that satisfy title keywords but fail hidden constraints.
> *When to apply*: When evaluating candidate products against multi-attribute instructions.
>
> **Successful child** ("Systematically Verify All Hidden Constraints", fitness=0.70, gen=1, `stable`):
> *Principle*: After selecting a candidate product, explicitly open the product description/features section and verify every constraint from the instruction that is not visible in the title or options.
> *When to apply*: When a candidate product matches title keywords but contains hidden attribute constraints.
>
> **Failed child** ("Systematically Verify Every Non-Visible Constraint", fitness=0.21, gen=1, `retired` at step 78):
> *Principle*: After selecting a product, scroll down and click on the 'Description' or 'Features' tab/section to read every specification.
> *When to apply*: Whenever a product page is opened.

The successful child reaches `stable` status, while the failed variant referenced interface elements such as "click on the Description tab," which is absent from WebShop; its fitness decayed rapidly and the retirement gate filtered it out at step 78.

E.4 ALFWorld: Mutation Outcomes

The initial skill "Pick Before You Wander" produced two mutations with opposite outcomes:

> **Parent** ("Pick Before You Wander", gen=0):
> *If the target object is visible or discovered, take it immediately; never leave it behind to explore other places.*

Successful child ("Verify, Grab, and Confirm Pickup", gen=1, fitness=0.70, `stable`):
When a needed object is visible, move directly to it, take it immediately, and then verify it is now in your inventory before proceeding.
→ Adds a **verification step** that catches pickup failures.

Failed child ("Retrieve Target Before Locating Desklamp", gen=1, fitness=0.24, `retired` at step 65):
First locate and pick up the target object specified in the goal before navigating to any furniture.
→ Introduces a rigid ordering that fails for tasks where the lamp must be found first.

From the same parent, mutation produced both a beneficial specialization (adding a verification step) and a harmful over-specification (imposing a rigid task ordering). The retirement mechanism correctly identified and removed the harmful variant.

E.5 Search-Augmented QA: Query Reformulation Lineage

A multi-generation forging chain on Search-Augmented QA:

Gen-0: "Disambiguation Add-Ons"
If the name is ambiguous, append clarifiers (birth year, nationality, known work) to the query.

↓ mutation

Gen-1: "Structured Query Escalation with Fallback" (fitness=0.21, `retired` at step 83)
Systematically escalate through three reformulation strategies when initial results lack evidence.
→ Too rigid; forced 3-step escalation even for simple queries.

↓ later mutation from same lineage

Gen-1: "Verify Before Acting" (fitness=0.70, `stable`)
Before performing any action on an entity, first confirm its identity by searching with disambiguating context.
→ Simpler, more general, and more effective.

Forging explores multiple directions from the same parent: the overly complex variant is retired while the simpler, more robust variant survives.

F KL Regularization under a Library-Conditioned Context

This appendix elaborates on how the KL penalty in Eq. (1) composes with the evolving skill library, complementing the non-stationarity of the fitness estimator formalized in Eq. (11).

Under our formulation (Eq. (2)), the library $\mathcal{S}^{(k)}$ enters the policy through the context p_t rather than through the policy parameters θ. The parameterization of π_θ is therefore invariant across forging cycles; what varies over training is the prompt distribution fed to the fixed network. The KL term $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\text{ref}})$ is always evaluated at the same context $c_{n,l}$ seen by both π_θ and π_{ref}, so the divergence is well-defined at every training step, regardless of how far $\mathcal{S}^{(k)}$ has drifted from $\mathcal{S}^{(0)}$.

Two structural properties align the fixed anchor π_{ref} with the library-conditioned inputs it encounters during RL. First, mutated skills are teacher-model rewrites of surviving parents that preserve the (v, b, ω) schema of Section 3.2, so new prompts differ from SFT-time prompts mainly in skill content rather than in structural form, on which π_{ref} remains a calibrated next-token predictor. Second, we use $\beta = 10^{-3}$ (Table 5), so the KL term acts as a soft regularizer against large single-step deviations rather than a tight anchor to the SFT policy's full conditional distribution. An alternative design is to re-anchor π_{ref} at each forging cycle, trading the clean separation between SFT initialization and RL optimization for tighter alignment between the reference policy and the current library; this variant is compatible with the rest of our framework, and the fixed-anchor choice is the simpler default we adopt here.

G Discussion

Design choices. Fitness is tracked as a cumulative running average (Eq. (6)), which integrates evidence over the full lifecycle of each skill. This is the natural estimator for our detection target, skills whose accumulated success record needs to be distinguished from skills that were never effective; it is what makes "once stable, now failing" a well-defined pattern to catch. A faster estimator such as an exponential moving average would respond more quickly to fitness changes, but this sensitivity cuts both ways: a skill that encounters a transiently hard batch of tasks would be flagged before the policy has stabilized, producing false retirements; empirically, the moving average lowers average success by 1.0 point (Table 14). The cumulative average's inertia is intentional—the lifecycle's usage-count gates (N_{retire}, N_{stable}) already guard against premature decisions; within that framework, the cumulative estimate reliably separates skills with a strong historical record from those that have consistently underperformed. Table 1 reports an average retirement-event lifespan of 38.0 training steps, showing that retirement occurs promptly relative to the 200-step training budget despite the conservative estimator.

Mutation survival. Across the three environments, 213 mutant skills were generated: 88 in ALFWorld, 55 in WebShop, and 70 in Search-Augmented QA. The majority survive the lifecycle: the mutant survival rate is 64.8%, 52.7%, and 62.9% respectively, so LLM-guided mutation produces viable skills more often than not. Each forging cycle requires up to K_{mutate} teacher model calls for mutation plus lightweight fitness bookkeeping.

Ablation library dynamics. In Table 3 the −mutation and −forging variants both retain the initial library size (44 on ALFWorld, 66 on WebShop, 41 on Search-Augmented QA). These numbers refer to $|\mathcal{S}_0|$, the distilled library *before* pre-retirement, to make cross-variant comparison direct. For −forging the entire forging method, including pre-retirement, runtime retirement, and mutation, is disabled, so the SFT data is unfiltered and no skills are added or removed after initialization. For −mutation, pre-retirement and runtime retirement both run, but the two mechanisms remove very few skills on these environments in practice: pre-retirement at $\delta_{\text{pre}} = 0.3$ with $N_{\min} = 3$ discards only initial skills that demonstrably hurt the base-model policy, and on the three environments this is a small fraction of $\mathcal{S}_0$ (typically 0–3 skills); the remaining skills then accumulate enough usage during RL to stay above δ_{retire}, so runtime retirement triggers rarely when mutation is off and no low-fitness children are being introduced. Size changes driven by the lifecycle therefore manifest primarily once mutation is re-enabled, which is consistent with the joint role of mutation and retirement reported in Section 5.3. The −pre-retirement variant confirms this picture from the opposite direction: with runtime retirement and mutation intact, the final library still settles at the cap (100, 95, 85) because the cap is the binding constraint once mutation is active. The 1.2–3.5% success drop therefore arises not from library size but from SFT data quality: pre-retirement is an integral first stage of the lifecycle that shapes the initial policy $\pi_{\theta_{\text{sft}}}$ on which the subsequent retirement and mutation operate.

The −retirement column in Table 3 reaches 132, 121, and 111 skills on the three environments, matching the *total-ever-created* count of the full method. This follows from the forging schedule: the number of forging cycles and the per-cycle budget K_{mutate} are fixed, so the cumulative number of child skills generated is largely invariant to whether low-fitness parents are subsequently removed. Retirement determines which skills remain in the active library at any point, not how many are produced during training; correspondingly the full method's active library settles at a much smaller size (100, 95, 85) while −retirement accumulates every generated skill.

Evaluation protocol. All reported numbers use stochastic decoding at temperature 1.0 for both training-time rollouts and test-time evaluation, so that evaluation noise reflects the policy's deployed behavior rather than a degenerate argmax. Each reported score is averaged over five evaluation runs of the best validation checkpoint with sampling seeds $\{30, 40, 42, 50, 62\}$. The mean $\pm$ standard deviation of the overall success rate is $92.4 \pm 0.5\%$ on ALFWorld, $78.4 \pm 0.4\%$ on WebShop, and $48.7 \pm 0.3\%$ on Search-Augmented QA, with the low variance confirming that reported results are stable under resampling.

Statistical significance. We test whether SKILLFORGE's overall success rate exceeds the strongest memory-augmented RL baseline, SkillRL, using a one-sided one-sample t-test against the SkillRL point estimate under the null $H_0 : \mu_{\text{SKILLFORGE}} = \mu_{\text{SkillRL}}$ with our five-seed evaluation sample ($n = 5$,

$\mathrm{df} = 4$). On ALFWorld ($t = 11.18$, $p = 1.8 \times 10^{-4}$), WebShop ($t = 31.86$, $p = 2.9 \times 10^{-6}$), and Search-Augmented QA ($t = 14.16$, $p = 7.2 \times 10^{-5}$), the improvement is significant at $p < 10^{-3}$ on all three environments. The corresponding two-sided 95% confidence intervals, $[91.78, 93.02]$, $[77.90, 78.90]$, and $[48.33, 49.07]$, do not overlap the SkillRL baseline values of 89.9, 72.7, and 46.8, so the 1.9–5.7% absolute gains are unlikely to be explained by sampling noise. Under the conservative assumption that SkillRL's per-seed variance matches our observed per-environment standard deviations of 0.5%, 0.4%, and 0.3% respectively, a pooled Welch's t-test yields $t = 7.91$ (ALFWorld), $t = 22.51$ (WebShop), and $t = 10.03$ (Search-Augmented QA), with $p < 10^{-4}$ on all three environments, preserving the conclusion.

Broader impact. Treating experience accumulation as a quality-control problem improves agent performance and yields interpretable artifacts: the SKILLFURNACE dataset supports the study of skill quality prediction and failure-aware generation. The collection contains only SFT trajectories, evolved skill libraries with fitness annotations, and retirement events with failure annotations, and introduces no risks beyond those inherent to LLM agents that act in interactive environments.

H Additional Controlled Experiments

This appendix reports controlled experiments that complement the ablation of Section 5.3. Unless stated otherwise, entries are success rate on ALFWorld and WebShop and accuracy on Search-Augmented QA (%), each the mean $\pm$ standard deviation over the five evaluation sampling seeds at the best validation checkpoint, following the protocol of Appendix G. **Avg** is the unweighted mean over the three environments. In every experiment all settings other than the one under study are held at their defaults, and differences smaller than the per-seed deviations are not read as an ordering.

Matched teacher budget. Among the lifecycle mechanisms only mutation uses the external teacher. We count the teacher rewrites B that SKILLFORGE performs over training, 88, 55, and 70 on ALFWorld, WebShop, and Search-Augmented QA, equal to the number of mutation children, and grant SkillRL the same B rewrites appended to its library with no lifecycle gate, that is, no `trial` admission and no fitness-driven retirement. Because SkillRL maintains no fitness scores, the B parents are sampled in proportion to retrieval frequency under SkillRL's own rollouts, so the control uses neither SKILLFORGE's fitness nor its lifecycle states. The gap between SKILLFORGE and the teacher-matched SkillRL measures the lifecycle at equal teacher budget, and the gap between the teacher-matched and the plain SkillRL measures distillation alone (Table 11). Averaged over the three environments, teacher distillation alone accounts for about 0.7 points and the lifecycle for about 2.7 points at a matched teacher budget. The lifecycle also keeps SKILLFORGE's library at the cap $S_{\max}$, whereas the teacher-matched SkillRL retains every skill it receives and ends with a larger, unpruned library. Mutation's small additive contribution in Table 3, 0.5 to 1.6 points, understates its role: retirement removes low-fitness skills, while mutation replenishes the library with improved variants so that pruning does not merely shrink coverage.

Table 11: Matched teacher budget. Rewrites: number of teacher rewrites; B is the number performed by SKILLFORGE in each environment.

Method	Rewrites	Lifecycle	ALFWorld	WebShop	Search-QA	Avg
SkillRL	0	no	89.9±0.5	72.7±0.4	46.8±0.3	69.8±0.2
SkillRL + teacher rewrites	B	no	90.3±0.5	74.2±0.5	47.0±0.3	70.5±0.3
SKILLFORGE	B	yes	**92.4**±0.5	**78.4**±0.4	**48.7**±0.3	**73.2**±0.2

Teacher capability. We vary only the mutation teacher $\mathcal{M}_T$. Besides the default Kimi-K2.5 we test two other capable instruction models, GLM-5.2 [50] and MiniMax M3 [51], and, as the strongest control, self-mutation, in which the base model itself serves as the teacher so that no external knowledge is injected. Disabling mutation while keeping retirement gives the floor (Table 12). Capable external teachers stay within 0.3 points of the default, while self-mutation falls back near the retirement-only floor and still exceeds SkillRL's 69.8 in Table 11.

Credit assignment. Eq. (7) attributes one binary reward to all K co-retrieved skills. We test this approximation in two ways (Table 13). The $K = 1$ stress test retrieves a single skill per episode,

Table 12: Teacher capability for LLM-guided mutation.

Mutation teacher $\mathcal{M}_T$	**ALFWorld**	**WebShop**	**Search-QA**	**Avg**
None (retirement only)	91.8±0.5	76.8±0.4	48.2±0.3	72.3±0.2
Base model (self-mutation)	91.5±0.5	77.1±0.5	48.0±0.4	72.2±0.3
GLM-5.2	92.3±0.4	77.9±0.4	48.5±0.3	72.9±0.2
MiniMax M3	92.5±0.5	78.1±0.4	48.9±0.3	73.2±0.2
Kimi-K2.5 (default)	92.4±0.5	78.4±0.4	48.7±0.3	73.2±0.2

so each reward is attributed exactly; it changes retrieval breadth and credit assignment together, so its success-rate gap is not by itself a clean measure of credit error. The per-skill attribution variant keeps the default $K = 6$, so the agent's context is unchanged, and replaces Eq. (7) with each skill's leave-one-out marginal contribution over its co-retrieval sets, which then drives all lifecycle transitions in place of shared fitness. For both we report the Spearman correlation of the resulting per-skill fitness ranking with the default and the agreement of retirement decisions; for per-skill attribution we also report the agreement of mutation-candidate selection.

Table 13: Credit assignment. ρ: Spearman correlation of the per-skill fitness ranking with the default shared reward of Eq. (7); Ret./Mut.: agreement of retirement and mutation-candidate decisions with the default. ALF./Web./QA: ALFWorld, WebShop, Search-Augmented QA.

Estimator	K	**ALF.**	**Web.**	**QA**	**Avg**	ρ	**Ret.**	**Mut.**
Shared (default)	6	92.4±0.5	78.4±0.4	48.7±0.3	73.2±0.2	–	–	–
Single skill	1	91.8±0.5	76.9±0.5	48.4±0.3	72.4±0.3	0.86	89%	–
Leave-one-out	6	92.7±0.5	77.9±0.5	48.8±0.4	73.1±0.3	0.91	93%	88%

Pooled over the three environments, per-skill attribution and the default reach the same retirement decision on 269 skills and different decisions on 21. The 21 disagreeing skills lie close to the threshold, with mean $|f_i - \delta_{\text{retire}}|$ of 0.07 against 0.31 for the agreeing ones, and are rarely used, with mean usage U_i of 23 against 48. Reassigning only these decisions within the default run, a targeted counterfactual rather than end-to-end retraining, changes aggregate success by $+0.1$. The shared-reward approximation therefore does not materially affect the lifecycle.

Design alternatives. Table 14 compares the main alternatives for three load-bearing design choices, whose rationale is given in Appendices B and G: the fitness estimator, the granularity of the lifecycle, and the weight used to sample mutation parents. Each alternative lowers average success relative to the default, most clearly the two-state lifecycle and the EMA, at 1.7 and 1.0 points of average success.

Table 14: Alternatives for three design choices of SKILLFORGE. EMA: exponential moving average; two states: `active` and `retired` only.

Design choice	**Variant**	**ALFWorld**	**WebShop**	**Search-QA**	**Avg**
Fitness	cumulative (default)	92.4±0.5	78.4±0.4	48.7±0.3	73.2±0.2
	EMA	91.1±0.5	77.2±0.5	48.4±0.3	72.2±0.3
Lifecycle	four states (default)	92.4±0.5	78.4±0.4	48.7±0.3	73.2±0.2
	two states	90.8±0.5	76.1±0.5	47.7±0.4	71.5±0.3
Mut. weight	$1 - f_i$ (default)	92.4±0.5	78.4±0.4	48.7±0.3	73.2±0.2
	$1/f_i$	91.9±0.5	77.3±0.5	48.5±0.4	72.6±0.3
	uniform	92.5±0.5	77.5±0.4	48.0±0.4	72.7±0.3

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (Section 1) state three contributions: fitness-driven lifecycle, retirement-aware training method with SkillFurnace dataset, and empirical results across three benchmarks. All are supported by experimental results in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 7 discusses the single training seed, episode-level credit assignment, model scale, teacher dependence, and the scope of SKILLFURNACE; Appendices G and H quantify these effects.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include formal theoretical results or proofs. The method is empirically validated.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 5.1 specifies the base model, training hyperparameters, and environment details. Appendix B provides the complete set of forging hyperparameters. Algorithm 1 gives the full forging cycle procedure.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code, training scripts, seed skill banks, and trained checkpoints are publicly released via `https://github.com/YuyaoGe/SkillForge` with setup and training instructions; all benchmarks used in the experiments are publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 and Appendix B describe environments, baselines, model, optimizer, learning rate, batch size, group size, skill library sizes, and GPU configuration. Appendix B lists all forging hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Standard deviations across five evaluation runs (sampling seeds at the best validation checkpoint) are reported in Appendix G (± 0.3–0.5% on the overall metric of each environment).

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5.1 and Appendix B specify compute resources (64× NVIDIA H200 GPUs, verl framework) and report the estimated wall-clock training cost and teacher-LLM call budget.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available benchmarks and does not involve human subjects or sensitive data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix G includes a broader impact statement. We do not foresee negative impacts beyond those inherent to LLM agent systems.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The released code and checkpoints target text-based research benchmarks (ALFWorld, WebShop, and question answering) and pose no high risk for misuse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All benchmarks (ALFWorld, WebShop, Search-R1) and the base model (Qwen2.5-7B-Instruct) are properly cited. These are publicly available research assets.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released code and checkpoints are documented in the repository README at `https://github.com/YuyaoGe/SkillForge` with setup, training, and evaluation instructions; the SKILLFURNACE dataset is described in Appendix C with statistics, schema, and failure taxonomy.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This work does not involve human subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The method uses an LLM teacher model for skill distillation and mutation (Section 4.3, Appendix B). The SkillFurnace failure categories in Section C are assigned by human annotators.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.