

SkillRubricsRL: Evolving Skill-Grounded Agents with Rubric-Guided Reinforcement Learning

Zihe Huang^{1,2*} Jianfei Pan^{2*} Yifan Zhang^{2*} Tao Chen² Jinghao Lin²
Kai Ruan² Xiucheng Xu¹ Jian Yao² Yijun He^{2†} Haihua Yang^{2†}

¹University of Chinese Academy of Sciences, Beijing, China ²ByteDance, Beijing, China

Abstract

Long-horizon LLM agents increasingly rely on retrieved procedural skills as external memory. However, existing skill-augmented RL frameworks typically supervise a bundle of retrieved skills only with trajectory-level outcome rewards, leaving two key questions unanswered: whether each skill was executed and whether it helped task completion. We formulate this as a per-skill credit assignment problem. To address it, we propose **SkillRubricsRL**, which pairs each skill with an applicability-aware trajectory-level rubric and derives two complementary signals: a fulfillment signal for training correct skill execution and a counterfactual utility signal for retrieval and skill-bank evolution. Across ALFWorld and WebShop, SkillRubricsRL reaches SOTA performance, with 97.7% ALFWorld success and 84.4% WebShop success, while improving the skill execution rate by over 50 percentage points. Without prompt-time skills, SkillRubricsRL achieves 97.7% success on ALFWorld and 82.8% on WebShop, remaining comparable to or better than its skill-on performance and demonstrating effective skill internalization. Code is available at <https://github.com/gminiheimdallr-a11y/SkillRubricsRL>.

1 Introduction

Modern LLM agents are increasingly trained not as standalone policies, but as policies coupled with external memory: skills, reflections, tool instructions, plans, and workflow templates that persist across episodes and accumulate with experience (Wang et al., 2023; Zhao et al., 2024; Wang et al., 2024b). In long-horizon interactive environments, such memory is no longer merely a prompt-time convenience; it becomes part of the agent’s capability surface. Reinforcement learning for these

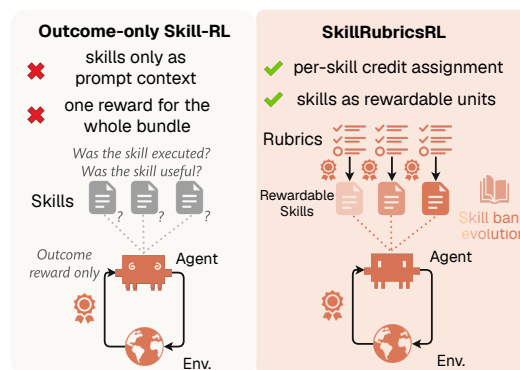


Figure 1: Conceptual overview of skill-level supervision in skill-augmented RL. Prior outcome-only methods use retrieved skills as prompt context and assign one scalar reward to the whole skill bundle, leaving skill execution and usefulness unobserved. SkillRubricsRL attaches rubrics to skills so that each skill can be evaluated as a rewardable unit, enabling per-skill credit assignment and skill-bank evolution.

agents is therefore not only about optimizing a policy, but about training a system that can retrieve, use, evaluate, and update its own external knowledge (Schulman et al., 2017; Guo et al., 2025). A rapidly growing line of work equips RL agents with reusable procedural skills: agents curate skills from experience, retrieve them at inference time, and expand the skill bank as new tasks or failure modes emerge (Wang et al., 2023; Zhang et al., 2026; Xia et al., 2026). However, in existing skill-augmented RL systems, retrieved skills are typically treated only as prompt context. During training, the environment provides only a trajectory-level outcome reward, so the system does not know **whether each retrieved skill was actually executed**, or **whether an executed skill was helpful**. Outcome-only RL thus supervises task completion, but leaves knowledge usage unsupervised. This creates a per-skill credit assignment problem: when multiple skills are surfaced into the context, a single scalar reward $r(\tau)$ cannot identify the marginal role of each skill, leaving the policy, retriever, and skill bank opti-

* Equal contribution.

† Co-corresponding authors.

mized against a signal blind to how knowledge is being used.

We audit this issue on ALFWorld using SkillRL as baseline, and find that the gap is substantial in practice. As shown in Fig. 2 (a), outcome-only RL achieves a task success rate of 89.9%, yet executes only 15.6% of the retrieved skills. Fig. 2 (b) shows a representative failure on the harder query "find two pan and put them in countertop". Several retrieved skills are off-target: they guide the agent toward cabinets and countertops, while the pans are located on stove burners. The policy follows these low-utility skills, neglects the relevant ones, and repeatedly moves among cabinets until exhausting the 30-step budget. This illustrates the core failure mode: without skill-level supervision, retrieval can mix useful and irrelevant skills, and the policy may still follow the wrong ones.

These observations suggest that a skill carries two distinct questions that outcome reward conflates: **Fulfillment**: does the agent behave in accordance with this skill? and **Utility**: does this skill contribute to task success? The two are not interchangeable: a skill can be faithfully followed but irrelevant, or useful in principle but ignored by the policy. Therefore, if a skill serves both as prompt guidance and as an object of RL supervision, it should carry its own evaluation contract. We instantiate this contract as an applicability-aware rubric attached to each skill, specifying when the skill is in scope, what behaviors fulfill it, and which behaviors violate it.

Building on this idea, we propose **SkillRubric-sRL**, a framework that pairs every skill with such a rubric and optimizes skill usage through two complementary signals. A rubric-based intrinsic estimator scores whether the trajectory fulfills each applicable skill, while a dropout-based counterfactual estimator measures whether removing a skill changes the final outcome. Together, these signals encourage the policy to execute relevant skills and guide the retriever and skill bank toward skills that actually improve task success. Across ALFWorld and WebShop, SkillRubricsRL surpasses prior state-of-the-art multi-skill-augmented RL baselines, improving task success by 2.8 percentage points (pp) to **97.7%** on ALFWorld and by 2.1 pp to **84.4%** on WebShop, while raising skill execution rate by more than 50 pp. Our contributions are summarized as follows:

- We identify per-skill credit assignment as a

structural bottleneck in skill-augmented RL: outcome-only rewards supervise task completion but leave skill usage unobserved, including which skills were executed and which actually helped.

- We propose SkillRubricsRL, which pairs each skill with an applicability-aware trajectory-level rubric and decomposes per-skill advantage into rubric-based fulfillment and dropout-based utility signals.
- On ALFWorld and WebShop, SkillRubricsRL improves both skill execution and task success over strong baselines; ablations show that fulfillment and utility supervision are complementary and both necessary.

2 Related Work

2.1 Skill-Augmented Agents

Skill-augmented LLM agents distill prior experience into retrievable procedural knowledge that is injected into the policy as prompt-time guidance (Wang et al., 2023; Zhao et al., 2024; Park et al., 2023). Early efforts such as Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024) convert episodic reflections into reusable verbal experience, while Mem0 (Chhikara et al., 2025) maintains a long-term retrievable event memory; more recent work including SkillRL (Xia et al., 2026), D2Skill (Tu et al., 2026), and Skill1 (Shi et al., 2026) organizes experience into structured skill banks that participate in reinforcement learning (Cheng et al., 2025; Wang et al., 2025). RetroAgent (Zhang et al., 2026) and related systems push further by introducing online skill updates and task-conditioned retrieval, giving the bank an in-training evolution loop. However, supervision in these methods remains anchored at terminal task success: the gradient never sees whether a retrieved skill was actually executed, ignored, or even harmful to the policy, conflating skill *fulfillment* with skill *utility*. In contrast, SkillRubricsRL endows every skill with a trajectory-level rubric that turns fulfillment into a trainable process reward, and uses counterfactual skill dropout to estimate per-skill utility independently of the policy gradient.

2.2 Rubrics and Process Reward

Beyond RLHF (Ouyang et al., 2022) and outcome-only reward models, a growing body of work seeks process-level supervision that provides denser and more interpretable signal for complex reasoning

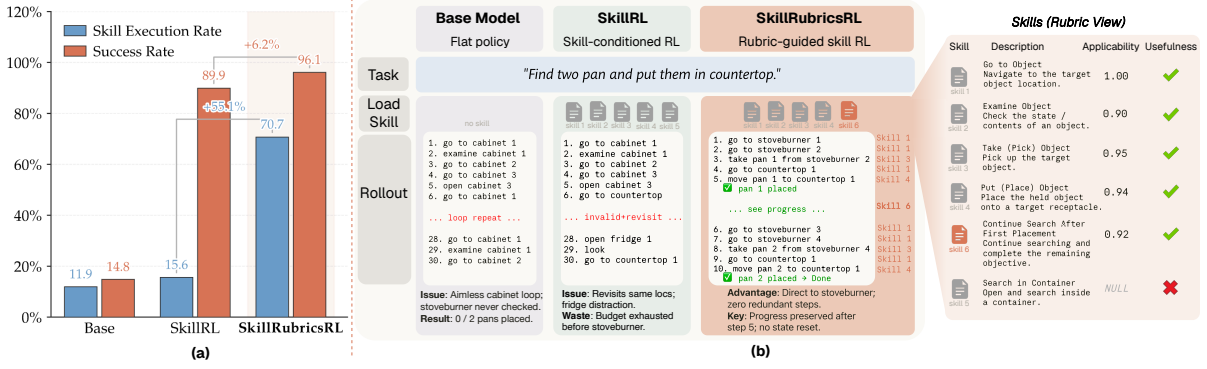


Figure 2: Qualitative example on a two-object ALFWorld task. The base model loops over irrelevant cabinets, while outcome-only SkillRL retrieves skills but follows low-utility guidance and exhausts the action budget. SkillRubricsRL retrieves skills whose rubric view exposes whether each skill is fulfilled and useful for the query, enabling the agent to preserve progress after placing the first pan and complete the remaining objective.

or long-horizon tasks (Uesato et al., 2022). Process Reward Models (PRMs) score individual reasoning steps and have shown more stable training dynamics than outcome reward in math and code domains (Wang et al., 2024a; Setlur et al., 2025). In parallel, rubric-based reward formulations constrain LLM judges with explicit, observable criteria, yielding more traceable and less hackable signals (Zheng et al., 2023; Gunjal et al., 2025). Unlike these formulations, which score a single response or a reasoning step, the rubrics in SkillRubricsRL operate over an action–observation trajectory and are bound one-to-one with skills in the bank, so that the prompt channel (what the agent is asked to do) and the reward channel (what the agent is rewarded for doing) cannot drift apart.

2.3 Agentic RL with Outcome Rewards

Reinforcement learning has become a dominant paradigm for training interactive LLM agents on long-horizon environments such as ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022), where PPO (Schulman et al., 2017) and grouped-advantage estimators such as RLOO (Ahmadian et al., 2024), GRPO (Shao et al., 2024), and GiGPO (Feng et al., 2026) are widely adopted. A shared bottleneck is sparse terminal reward over long trajectories, which complicates credit assignment and has motivated a variety of auxiliary or intrinsic reward designs (Bai et al., 2024). However, curiosity-, self-consistency-, or verifier-style auxiliary rewards are typically opaque scalars that resist behavior-level attribution, in contrast to our rubric criteria, each of which is bound to an observable trajectory-level behavior. SkillRubricsRL unifies these three lines—skill-

augmented agents, process-level rubric reward, and agentic RL—into a framework, and uses counterfactual skill dropout to decouple skill *fulfillment* from skill *utility* so that the bank can be trained, audited, and evolved over the course of learning.

3 SkillRubricsRL

Given a task instruction x , a retriever selects a skill subset $\mathcal{S}_x$ from the skill bank $\mathcal{S}$ and inserts their principles into the policy prompt. The policy π_θ then produces a trajectory τ with outcome reward $r_{\text{env}}(\tau)$. SkillRubricsRL augments this loop with two skill-level signals: fulfillment, which evaluates whether retrieved skills are executed, and utility, which estimates whether they improve task outcomes. To obtain these signals, each skill is paired with an applicability-aware trajectory-level rubric for post-rollout evaluation, while stochastic skill replacement provides counterfactual evidence for retrieval and skill-bank evolution. Figure 3 summarizes the workflow.

3.1 Skills as Dual-Use Objects

SkillRubricsRL represents each skill with two synchronized views. The *prompt view* is a natural-language principle that guides the policy during rollout. The *reward view* is a rubric that evaluates, after rollout, whether the trajectory satisfied the behavioral requirements of that skill.

Formally, each skill $s \in \mathcal{S}$ is a tuple

$$s = (\text{id}_s, t_s, p_s, a_s, \mathcal{R}(s)), \quad (1)$$

where id_s is the skill identifier, t_s is a title, p_s is the prompt principle, a_s is an application condition, and $\mathcal{R}(s)$ is the rubric. The fields (t_s, p_s, a_s)

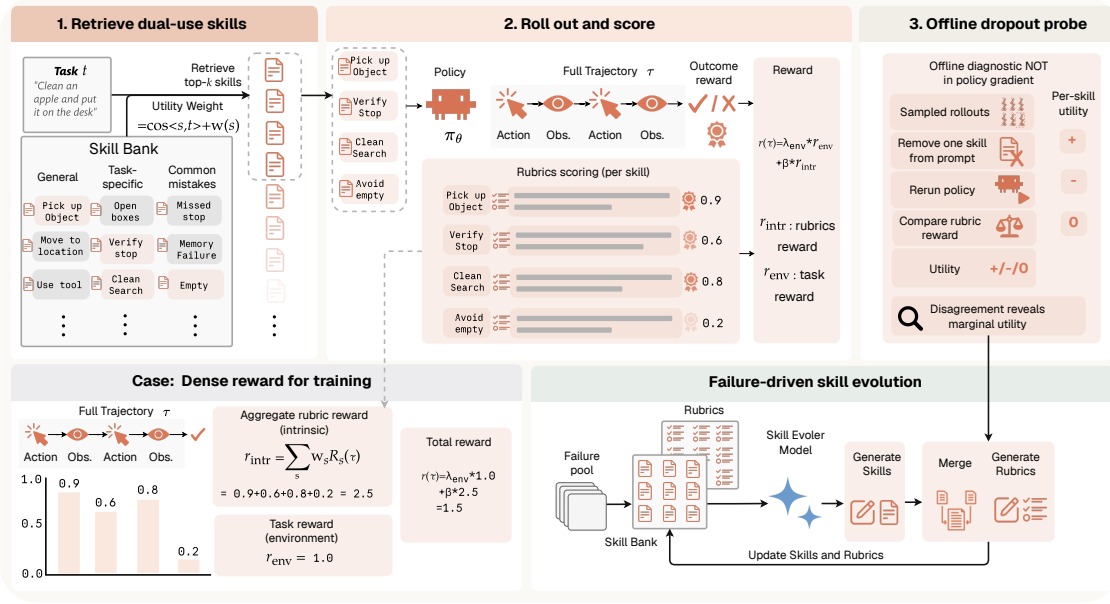


Figure 3: The training workflow of SkillRubricsRL. Retrieved skills are used as dual-use objects: their principles guide policy rollout, while their rubrics provide trajectory-level fulfillment rewards. Stochastic skill replacement estimates per-skill utility, which updates retrieval and skill-bank evolution.

define when and how the skill should guide behavior, while $\mathcal{R}(s)$ defines how the resulting trajectory should be evaluated.

The rubric is a weighted set of observable criteria:

$$\mathcal{R}(s) = \{(c_j, w_j)\}_{j=1}^m, \quad (2)$$

where each criterion c_j can be judged from a complete action–observation transcript and w_j is its weight. The application condition a_s makes the rubric applicability-aware: a skill is rewarded only when it is in scope for the task and trajectory. Rubrics are generated from the same skill specification used in the prompt, so policy guidance and reward evaluation are aligned.

3.2 Rubric-Grounded Fulfillment Reward

Rubric scoring provides the first per-skill signal: *fulfillment*. After an episode terminates, the trajectory collector assembles the full transcript z_τ . For each retrieved skill $s \in \mathcal{S}_x$, an evaluator $q(\cdot, \cdot) \in [0, 1]$ scores the transcript against every rubric criterion. The per-skill fulfillment score is

$$r_{\text{ful}}(\tau, s) = \frac{\sum_j w_j q(z_\tau, c_j)}{\sum_j w_j}. \quad (3)$$

Aggregating over retrieved applicable skills gives the intrinsic fulfillment reward

$$r_{\text{intr}}(\tau) = \frac{1}{|\mathcal{S}_x|} \sum_{s \in \mathcal{S}_x} r_{\text{ful}}(\tau, s). \quad (4)$$

The policy is trained with a reward that combines task success and skill fulfillment:

$$r(\tau) = \lambda_{\text{env}} r_{\text{env}}(\tau) + \beta r_{\text{intr}}(\tau). \quad (5)$$

This reward keeps environment success as the primary objective while providing dense supervision for whether retrieved skills are actually executed. Because fulfillment scores remain indexed by skill before aggregation, trajectories with the same outcome reward but different skill usage patterns become distinguishable during training and analysis.

3.3 Counterfactual Utility via Skill Replacement

Fulfillment alone is insufficient: a skill can be faithfully followed but irrelevant, redundant, or harmful. SKILLRUBRICSRL therefore estimates a second signal, utility, which measures whether a skill improves task outcomes.

For a task context t and retrieved set $\mathcal{S}_t$, we create perturbed prompts by stochastically replacing selected skills while keeping the task and policy fixed. Let $\mathcal{S}_t^{(-s)}$ denote a perturbed skill set in which skill s is replaced by an alternative skill. We compare rollouts under the original and perturbed skill sets and estimate the utility of s as the outcome difference

$$\Delta(s, t) = \mathbb{E}[r_{\text{env}}(\tau) \mid \mathcal{S}_t] - \mathbb{E}[r_{\text{env}}(\tau) \mid \mathcal{S}_t^{(-s)}]. \quad (6)$$

A positive $\Delta(s, t)$ indicates a helpful skill; a non-positive value indicates a redundant or harmful one. Rather than entering the policy reward, this signal updates retrieval and the skill bank: we aggregate $\Delta(s, t)$ via EMA, $w_{n+1}(s) = (1 - \eta)w_n(s) + \eta\mathbb{E}_t[\Delta(s, t)]$, rank candidates by $\cos(\mathbf{s}, \mathbf{t}) + \alpha w_n(s)$, and archive skills with $w_n(s) < \tau$, where n indexes skill-weight updates and t denotes task contexts.

3.4 Retrieval and Skill-Bank Evolution

Retrieval links policy guidance and skill evaluation: selected skills are inserted into prompt and scored by their paired rubrics. Utility estimates update retrieval by promoting helpful skills and down-weighting unused, redundant, or harmful ones.

The skill bank is evolved from failed trajectories containing the task, task type, and action–observation transcript. An LLM updater proposes general and task-specific skills, which are filtered by per-update budgets, deduplicated by title and embedding similarity, and rewritten when they encode brittle object–location priors. Remaining candidates are compared with the active bank; exact title matches are treated as updates, and semantically similar skills are merged only when an LLM verifies that they share the same objective, trigger condition, task phase, and required behavior.

Accepted skills are added to the active bank, duplicates are archived, and merged skills either update an existing entry or create a fresh entry with parent ids. After each update, retrieval embeddings are invalidated and rubrics are generated for newly admitted skills, making them immediately usable for both prompting and reward computation.

Algorithm 1 summarizes the full training loop.

4 Experiments

4.1 Experiment Setup

Our experiments evaluate whether explicit skill-level supervision improves task performance and knowledge usage. We focus on three questions.

Q1: Does making skills rewardable improve long-horizon task success? This is tested by comparing SkillRubricsRL against prompting, memory-only, RL-only, and skill-augmented RL baselines. **Q2: Do fulfillment and utility signals matter separately?** This is tested through ablations that remove rubric reward, skill-dropout utility, contribution-aware retrieval, or skill evolution. **Q3: Does the learned policy internalize procedural**

Algorithm 1 SkillRubricsRL: Rubric-Grounded Skill-Augmented RL

Require: policy π_θ , environment $\mathcal{E}$, tasks $\mathcal{D}$, skill bank $\mathcal{S}$, rubric bank $\mathcal{R}$, group size G , update interval K , weights $\lambda_{\text{env}}, \beta$.
Ensure: trained policy π_{θ^*} , evolved skill bank $\mathcal{S}^*$ and rubric bank $\mathcal{R}^*$.
1: Initialize $\mathcal{S}$ and generate $\mathcal{R}(s)$ for skills without rubrics.
2: **for** training step $t = 1, \dots, T$ **do**
3: Sample task batch $B \subset \mathcal{D}$.
4: Retrieve skill set $\mathcal{S}_x$ for each task $x \in B$ and insert principles $\{p_s : s \in \mathcal{S}_x\}$ into prompts.
5: Roll out π_θ to collect trajectories $\mathcal{T}$.
6: **for all** $\tau \in \mathcal{T}$ **do**
7: Score z_τ against $\mathcal{R}(s)$ for each retrieved skill $s \in \mathcal{S}_x$.
8: Compute $r_{\text{intr}}(\tau)$ and combined reward $r(\tau)$ by Eq. 5.
9: **end for**
10: Update π_θ with grouped advantages over environment and fulfillment rewards.
11: Stochastically replace selected skills and estimate utility $\Delta(s, x)$.
12: Update skill weights and retrieval ranking using utility estimates.
13: **if** $t \bmod K = 0$ **then**
14: Analyze recent failure traces to propose candidate skills.
15: Filter duplicates, merge near-duplicates, admit useful skills, and generate rubrics.
16: **end if**
17: **end for**
18: **return** $\pi_{\theta^*}, \mathcal{S}^*, \mathcal{R}^*$.

Table 1: Training procedure of SkillRubricsRL.

skills? This is tested by comparing skill-on evaluation, where retrieved skills remain in the prompt, with skill-off evaluation, where prompt-time skills are removed while keeping trained policy fixed.

Benchmark. We evaluate on two long-horizon interactive benchmarks. ALFWorld requires embodied household reasoning across pick, look, clean, heat, cool, and pick-two-object tasks. Many of its skills correspond to observable action-order constraints, making it a direct test of rubric-grounded fulfillment supervision. WebShop requires shopping decisions under user constraints, where retrieved skills can help only if they are useful for the current product and attribute requirements; it therefore stresses utility-aware retrieval.

Baseline. We compare against groups of methods. The first contains prompting and memory-only agents without RL training, including Zero-Shot, ReAct, Reflexion, Mem0, and ExpeL. The second contains RL-trained agents without skill credit, including PPO, RLOO, GRPO, and GiGPO. The third contains skill- or memory-augmented RL agents, including EvolveR, Mem0 w/ GRPO, SimpleMem w/ GRPO, SkillRL, Skill0, D2Skill, RetroAgent, and Skill1. These are the closest baselines because they use external procedural knowledge, but they do not explicitly supervise whether retrieved skills are fulfilled or useful.

For ALFWorld, we report per-category success rate and the average across categories. For WebShop, we report continuous score and binary success rate. During training, early rollouts use retrieved skills to bootstrap skill-conditioned behavior; after 75 training steps, each rollout batch is

Method	ALFWorld (Success Rate, %)							WebShop	
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	Score	Succ.
<i>w/o training</i>									
Zero-Shot (Yang et al., 2024)	33.4	21.6	19.3	6.9	2.8	3.2	14.8	26.4	7.8
ReAct (Yao et al., 2023)	48.5	35.4	34.3	13.2	18.2	17.6	31.2	46.2	19.5
Reflexion (Shinn et al., 2023)	62.0	41.6	44.9	30.9	36.3	23.8	42.7	58.1	28.8
Mem0 (Chhikara et al., 2025)	54.0	55.0	26.9	36.4	20.8	7.7	33.6	23.9	2.0
ExpeL (Zhao et al., 2023)	21.0	67.0	55.0	52.0	71.0	6.0	46.3	30.9	11.2
<i>RL w/o skills</i>									
PPO (Schulman et al., 2017)	92.3	64.0	92.5	89.5	80.3	68.8	80.4	81.4	68.7
RLOO (Ahmadian et al., 2024)	87.6	78.2	87.3	81.3	71.9	48.9	75.5	80.3	65.7
GRPO (Shao et al., 2024)	90.8	66.1	89.3	74.7	72.5	64.7	77.6	79.3	66.1
GiGPO (Feng et al., 2026)	97.7	82.7	98.8	83.7	89.3	79.2	90.8	84.4	72.8
<i>Single-skill RL agents</i>									
SKILL0 (Lu et al., 2026)	100.0	85.8	94.6	81.9	85.7	80.1	89.8	85.1	74.2
Skill1 (Shi et al., 2026)	100.0	<u>98.6</u>	97.3	<u>99.2</u>	96.1	<u>96.0</u>	<u>97.5</u>	89.7	<u>82.9</u>
<i>Multi-skill RL agents</i>									
EvolveR (Wu et al., 2025)	64.9	33.3	46.4	13.3	33.3	33.3	43.8	42.5	17.6
Mem0 (Chhikara et al., 2025) w/ GRPO	78.1	54.8	56.1	31.0	65.0	26.9	54.7	58.1	37.5
SimpleMem (Liu et al., 2026) w/ GRPO	89.5	36.3	60.0	50.0	64.9	26.3	62.5	67.8	46.9
SkillRL (Xia et al., 2026)	97.9	71.4	90.0	90.0	95.5	87.5	89.9	85.2	72.7
D2Skill (Tu et al., 2026)	97.1	100.0	75.0	87.5	100.0	78.6	90.6	<u>91.1</u>	80.5
RetroAgent (Zhang et al., 2026)	97.9	90.9	99.2	92.9	85.3	91.0	94.9	88.9	82.3
SkillRubricsRL (skill-on)	100.0	100.0	100.0	77.9	<u>96.2</u>	100.0	96.1	88.5	84.4
SkillRubricsRL (skill-off)	100.0	92.9	96.7	100.0	95.0	100.0	97.7	92.7	82.8

Table 2: Main results on ALFWorld and WebShop, organized by training and skill usage. Bold marks the best result in each column, and underline marks the second-best result. Avg. denotes average success rate; Succ. denotes WebShop success rate.

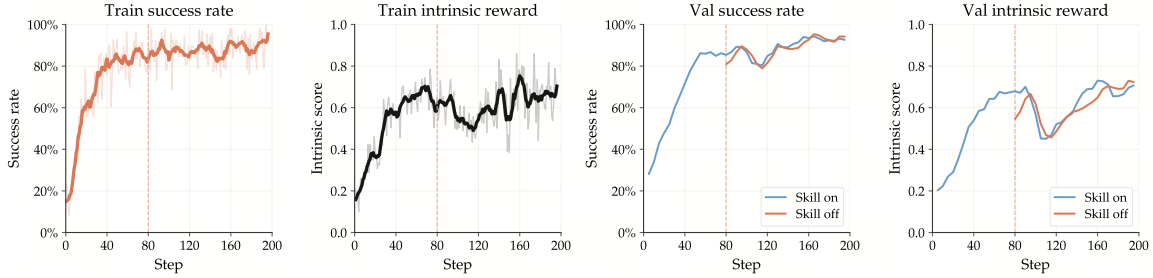


Figure 4: Training and test dynamics of SkillRubricsRL on ALFWorld. The left two panels show training success rate and intrinsic reward; the right two panels show test success rate and intrinsic reward. The dashed line marks step 75, after which training rollouts are split evenly between skill-on and skill-off prompts, with intrinsic rubric rewards computed for both. Evaluation is reported separately under skill-on and skill-off settings, where skill-off removes prompt-time skills to test procedural internalization.

split evenly between skill-on prompts and skill-off prompts. This mixed training forces the policy to benefit from external skills when available while learning to execute reusable procedures without prompt-time skill text. Table 2 measures task performance, Table 3 isolates the contribution of each component, and Figures 4–5 show whether reward and validation dynamics support our mechanism.

4.2 Main Results

Task performance. Table 2 shows that explicit skill-level evaluation improves over strong skill-augmented RL baselines. Prompting and memory-only methods provide useful context but do not optimize the policy, while RL-only methods learn long-horizon behavior but remain skill-blind. Multi-skill RL baselines perform better, confirming that procedural memory is valuable, but their retrieved skills

are still supervised only through trajectory-level rewards. SkillRubricsRL improves this setting by making each retrieved skill rewardable and by using utility estimates to refine retrieval and the skill bank. It achieves the best WebShop success rate with skill-on evaluation and the best ALFWorld average with skill-off evaluation, improving over the strongest prior multi-skill baseline by 2.8 percentage points on ALFWorld average success and 2.1 percentage points on WebShop success.

Skill-on vs. skill-off: external use and internalization. The skill-on/off comparison is the key diagnostic for whether the method merely relies on prompt-time skills or transfers procedural knowledge into the policy. In the skill-on setting, retrieved skill principles are visible at inference time, so performance measures the combined effect of

Table 3: Ablation study on ALFWorld. Each row removes one mechanism from SkillRubricsRL: removing the rubric reward tests whether per-skill execution credit is necessary; removing the skill-dropout utility tests whether execution and contribution must be separated and removing skill evolution freezes the skill bank. Subscript values next to “Avg.” indicate the percentage point difference from the best average success rate in the table.

Variant	ALFWorld Success Rate (%)						
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.
SkillRubricsRL (skill-on)	100.0	100.0	100.0	77.9	96.2	100.0	96.1 (−1.6)
SkillRubricsRL (skill-off)	100.0	92.9	96.7	100.0	95.0	100.0	97.7 (+0.0)
w/o rubric reward	96.9	71.3	95.4	91.7	88.9	92.2	91.4 (−6.3)
w/o skill-dropout utility	100.0	85.0	100.0	92.6	78.3	100.0	94.5 (−3.2)
w/o skill evolution	100.0	67.9	91.0	100.0	83.7	63.9	87.5 (−10.2)

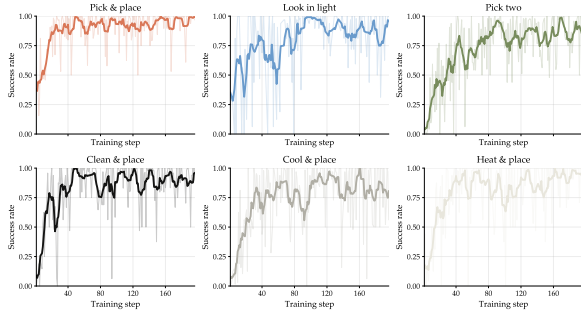


Figure 5: Training success across ALFWorld task families. All categories improve over training, while compositional tasks show fluctuations, highlighting the need for task-specific skills and rubric-based procedural supervision in multi-step settings.

the learned policy and the external skill bank. In the skill-off setting, these principles are removed while the trained policy is unchanged, so retained performance reflects internalized procedures. The 50/50 skill-on/off rollout mixture after step 75 makes this test meaningful: the policy is explicitly trained both to use external skills and to act without them.

The results show both effects. On ALFWorld, skill-off reaches 97.7% average success, exceeding the skill-on average of 96.1%. This indicates that many household procedures have been internalized: after rubric-grounded training with mixed skill availability, the policy can execute reusable behaviors such as valid first actions, systematic navigation, object acquisition, and required task phases even without retrieved skill text. This is important because it shows that rubrics are not only evaluating prompt following; they shape the policy into a more capable standalone actor. On WebShop, skill-on obtains the highest success rate, 84.4%, while skill-off remains competitive at 82.8%. This suggests that the policy internalizes general decision routines, but prompt-time skills remain useful for instance-specific constraints such as product attributes and purchase requirements.

Why the gap matters. The difference between skill-on and skill-off illustrates why fulfillment and utility must be separated. If skill-on is better, the external skill bank provides useful task-specific guidance at inference time. If skill-off is better or comparable, the policy has internalized the useful parts of the skill bank and may avoid over-constraining or mismatched retrieved skills. Thus, the goal is not simply to maximize dependence on retrieved text, but to train an agent that can both use external skills when they help and retain reusable procedures when absent. This directly supports the central claim of SkillRubricsRL: explicit skill-level evaluation turns procedural memory from passive context into training signal.

Process evidence from training dynamics. Figure 4 provides mechanism-level evidence on ALFWorld. During training, task success and intrinsic rubric reward improve, indicating that the policy is not only optimizing sparse terminal reward but also learning to satisfy retrieved skill rubrics. After step 75, the mixed skill-on/off rollout regime preserves this improvement under both prompt conditions, supporting procedural internalization.

Figure 5 further breaks down training success by ALFWorld task family. All six categories improve over training, but the more compositional families, such as pick-two-object, heat-and-place, cool-and-place, and clean-and-place, show larger fluctuations. This pattern is expected: these tasks require ordered subgoals, state tracking, and repeated object–receptacle interactions, so a single outcome reward provides weak process feedback. The per-category curves therefore motivate the task-specific part of the skill bank and the use of rubric criteria that reward intermediate procedural fulfillment rather than only final success.

4.3 Ablations

Table 3 isolates the components of SkillRubricsRL. Removing rubric reward causes the average success rate to drop from 97.7% to 91.4%, showing that trajectory-level outcome reward alone is not enough to teach reliable skill execution. The largest drops occur on categories such as look and cool, where success depends on following a multi-step procedure rather than merely reaching a terminal state. This supports the role of fulfillment supervision: rubrics convert retrieved skills into dense, observable execution targets.

The utility-related ablations show that correct execution alone is insufficient. Removing skill-dropout utility lowers average success to 94.5%, and removing contribution-aware retrieval drops it to 94.3%. In both variants, the policy can still receive rubric reward, but the system loses the ability to distinguish useful skills from redundant or harmful ones under multi-skill retrieval. This confirms that utility estimation is needed to promote only fulfilled skills that benefit future retrieval.

Freezing skill evolution produces the largest degradation, reducing average success to 87.5%. This indicates that initial skill bank does not cover all failure modes encountered during RL training. Dynamic evolution allows the system to turn repeated failures into new or refined skills, filter brittle object–location priors, merge redundant candidates, and attach rubrics to admitted skills. The ablation therefore supports the full loop proposed in the method section: fulfillment trains execution, utility shapes retrieval, and skill evolution expands and cleans the procedural memory over time.

4.4 Case Study

Fig. 2 shows a two-object ALFWorld task where the agent must find two pans and place them on a countertop. The base model repeatedly searches cabinets and fails to progress. Outcome-only SkillRL retrieves procedural skills, but several provide low-utility guidance for this instance: they bias the agent toward cabinets and countertops even though the pans are on stove burners. Since the final reward is zero, this baseline cannot determine whether the failure comes from ignoring a useful skill, following an irrelevant one, or lacking the right search procedure.

SkillRubricsRL makes this failure mode actionable. Each retrieved skill is paired with a rubric that checks procedural events, such as searching plausi-

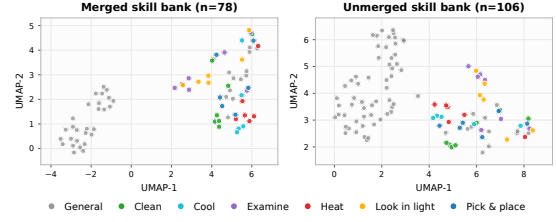


Figure 6: UMAP visualization of the ALFWorld skill bank before and after releasing recorded merge operations. The merged bank is more compact while preserving broad semantic coverage.

ble receptacles, acquiring a target object, returning to the destination, and repeating the acquire–place cycle for the required count. In this case, the agent first finds and places one pan, then preserves that progress instead of restarting the search or looping over the same cabinets. The utility estimator distinguishes fulfilled but unhelpful skills from skills that improve the trajectory, allowing low-utility guidance to be down-weighted and missing useful behaviors to be added to the skill bank. This illustrates the full credit loop: external skills guide the policy, rubrics make their execution observable, and utility estimates decide which skills should shape future retrieval and training.

4.5 Skill Merging Analysis

We analyze skill merging on the filtered final skill bank. During evolution, many generated candidates are discarded before entering memory because they are overly specific, brittle, redundant, or not useful for the current task distribution. Thus, these statistics reflect merging after candidate filtering rather than the total number of generated skills.

In the final ALFWorld bank, SkillRubricsRL records 28 merge operations. Releasing all recorded merges expands the active bank from 78 to 106 skills, with the increase mainly coming from general-purpose skills. As shown in Fig 6, merged bank is more compact while still spanning the main semantic regions of the unmerged bank. This suggests that merging primarily consolidates fine-grained variants of reusable general procedures, instead of removing task-specific coverage. Detailed examples are provided in Appendix Tab 8.

5 Conclusion

The key bottleneck in skill-augmented RL is not storing skills, but making training use them effectively. With multi-skill retrieval, trajectory-level

reward cannot identify which skills were executed or which ones helped. SkillRubricsRL addresses this with a dual-use skill representation: a prompt view guides the policy, while a reward view enables rubric-grounded fulfillment evaluation. We further estimate skill utility through stochastic replacement, separating execution from contribution. On ALFWorld and WebShop, our method outperforms strong multi-skill baselines.

Limitations

Our work has several limitations. First, we evaluate SkillRubricsRL on two long-horizon interactive benchmarks (ALFWorld and WebShop). While these benchmarks jointly cover embodied reasoning and web-based decision making, we believe evaluation on an even broader set of environments would further strengthen the generality of our conclusions, which we view as a promising direction for follow-up studies. Second, the rubric formulation in this paper is instantiated for procedural skills in interactive environments; we see it as a natural opportunity to explore how the same framework can be extended to richer skill schemas, such as declarative or compositional skills.

References

- Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267.
- Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495.
- Mingyue Cheng, Jie Ouyang, Shuo Yu, Ruiran Yan, Yucong Luo, Zirui Liu, Daoyu Wang, Qi Liu, and Enhong Chen. 2025. Agent-r1: Training powerful llm agents with end-to-end reinforcement learning. *arXiv preprint arXiv:2511.14460*.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2026. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408.
- Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*.
- Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.
- Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2025. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pages 60808–60838.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and

- An Zhang. 2026. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.
- Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, and Dongbin Zhao. 2026. Dynamic dual-granularity skill bank for agentic rl. *arXiv preprint arXiv:2603.28716*.
- Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2024a. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*.
- Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, and 1 others. 2025. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.
- Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xueming Yang, Yufan Shen, Yuxin Wang, and 1 others. 2025. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*.
- Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, and 1 others. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.
- Xiaoying Zhang, Zichen Liu, Yipeng Zhang, Xia Hu, and Wenqi Shao. 2026. Retroagent: From solving to evolving via retrospective dual intrinsic feedback. *arXiv preprint arXiv:2603.08561*.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*.

A Appendix

A.1 Additional Results

A.1.1 Case Study: How Skills Evolve

To make the dynamic memory interpretable, we summarize the learned skill bank and present three concrete cases: semantic skill merging, effective generation of new task-specific skills, and the final evolution path of currently used dynamic skills.

Skill categories. The final ALFWorld skill bank contains both hand-initialized skills and dynamically introduced skills. Although many active skills are broad heuristics, dynamically generated skills are concentrated in actionable failure modes exposed by RL training. In particular, most dynamic additions address first-step stalling and invalid or empty actions, while a smaller set captures task-specific procedural constraints such as cleaning, heating, light-source use, and quantity grounding.

Per-task training dynamics. Figure 5 complements the aggregate training curves in the main text by breaking ALFWorld success down by task family. The curves show that SkillRubricsRL improves across all six categories, but the learning profiles differ by procedural structure. Pick-and-place and heat-and-place rise quickly, while look-in-light, cool-and-place, and pick-two-object tasks exhibit larger variance because they require additional grounding steps such as locating a light source, preserving object state after cooling, or tracking multiple target instances. This supports the role of task-specific skills and rubrics: the model receives dense feedback for category-specific procedures rather than relying only on a single terminal success signal.

Case 1: semantic merging of redundant skills. When the LLM repeatedly proposes skills for the same failure mode, the updater does not simply append every candidate. Instead, it merges semantically similar candidates with existing skills and archives superseded versions. This is visible in the *Immediate First Navigation* chain, where multiple variants of a first-action anti-stalling rule are consolidated into a more precise active skill.

Representative merge chain. dyn_002 $\rightarrow$ dyn_005 $\rightarrow$ dyn_009 $\rightarrow$ dyn_013 $\rightarrow$ dyn_016 $\rightarrow$ dyn_040 $\rightarrow$ dyn_048 $\rightarrow$ dyn_058 $\rightarrow$ dyn_066 $\rightarrow$ dyn_094 $\rightarrow$ dyn_110.

Case 2: effective task-specific skill generation. Besides generic anti-stalling rules, the updater gen-

erates task-specific procedural skills. These skills are useful because ALFWorld tasks often require a fixed latent sequence, e.g., find-clean-place or find-heat-place. The examples below show generated skills that encode such constraints explicitly.

Case 3: current dynamic skills as evolved memory. The active dynamic skills in the final bank are not isolated one-shot additions. Several are endpoints of repeated proposal-merge-archive cycles. The final skill therefore reflects a compressed history of recurring failures. For example, first-step failures repeatedly produced candidates such as *Break Start Paralysis*, *Mandatory Initial Navigation*, and *Format Initial Go Command*; these were merged into successive versions of an active first-navigation skill.

Takeaway These cases show that the dynamic skill bank is not merely an ever-growing list. It behaves as a curated memory: redundant proposals are merged, failed behaviors are converted into actionable rules, and active skills become increasingly specific about when and how the policy should act.

A.2 Human Validation

We sampled 60 trajectory-skill pairs from complete interaction trajectories in ALFWorld and WebShop. Three expert annotators independently labeled the same set of 60 samples. Each sample included the task instruction, the skill principle and its applicability condition, the corresponding rubric criteria, and the complete action-observation trajectory. Without access to the LLM Judge outputs, the annotators assessed skill applicability, the degree to which each criterion was satisfied, and whether the skill’s core procedure was actually executed.

We aggregated the human annotations into consensus labels and compared them with the Judge’s skill-execution decisions and rubric scores. Because a single trajectory may correspond to multiple skill pairs, all confidence intervals were computed using a trajectory-clustered bootstrap. This avoids treating samples derived from the same trajectory as independent.

The annotators achieve κ values of 0.601 and 0.671 for skill applicability and rubric criteria, respectively, indicating that the rubric can be applied with reasonable consistency. Against the human consensus, the Judge achieves strong agreement on skill execution (Cohen’s $\kappa = 0.776$; $F_1 = 0.928$). Its continuous rubric scores are also

Current skill	Evolution source	Title	Final behavior encoded
dyn_110	long merge chain	Immediate First Navigation	Upon starting a task, immediately select a visible receptacle or furniture item exactly as it appears in the initial observation and explicitly issue a 'go to [target]' command to begin exploration. Never return empty text, skip your turn...
dyn_032	active dynamic bank	Prevent Empty Action Strings	Never submit a blank or empty action string. Always formulate a concrete, syntactically valid command (e.g., 'go to [receptacle]') to ensure the simulation advances.
dyn_062	active dynamic bank	Enforce Valid Action Syntax	Always construct your action using valid ALF-World command syntax and environment verbs (e.g., go to, take, put, open, close). Never submit an empty string or thought-only output.
dyn_090	active dynamic bank	Identify Source and Destination	Deconstruct the task to separate the target object from its final destination, and immediately execute a 'go to' command to begin the initial object search.
dyn_091	active dynamic bank	Establish Cleaning Sequence	Recognize that the object must be found, taken to a sinkbasin, cleaned, and then placed, and start immediately by navigating to a receptacle to find the target object.
dyn_093	active dynamic bank	Plan Appliance Sequence	Note the requirement to use a microwave for heating, and immediately issue a navigation command to begin searching for the target object to be heated.

Table 4: Current dynamic skills as endpoints of skill-bank evolution. The final active bank retains compact rules that summarize repeated training failures.

Task type	Skill ID	Title	Principle	Why it helps
clean	dyn_091	Establish Cleaning Sequence	Recognize that the object must be found, taken to a sinkbasin, cleaned, and then placed, and start immediately by navigating to a receptacle to find the target object.	Encodes the required find-clean-place phase order.
heat	dyn_093	Plan Appliance Sequence	Note the requirement to use a microwave for heating, and immediately issue a navigation command to begin searching for the target object to be heated.	Encodes appliance use before final placement.
look_at_obj_in_light	dyn_092	Acknowledge Light Requirement	Identify the required light source (e.g., desk lamp) from the task, and begin by navigating to find the target object before attempting to approach the light.	Makes the light-source constraint explicit.
pick_and_place	dyn_090	Identify Source and Destination	Deconstruct the task to separate the target object from its final destination, and immediately execute a 'go to' command to begin the initial object search.	Separates source object from destination receptacle.
examine	dyn_089	Address Target Quantities	If the task specifies finding multiple instances of an object (e.g., 'two spraybottles'), immediately initiate the search for the first instance by moving to a visible...	Prevents stopping after finding only one required instance.
general	dyn_062	Enforce Valid Action Syntax	Always construct your action using valid ALFWorld command syntax and environment verbs (e.g., go to, take, put, open, close). Never submit an empty string or thought-o...	Prevents blank actions and keeps interaction moving.

Table 5: Examples of effective dynamically generated skills. These skills convert repeated failure patterns into explicit procedural guidance for future rollouts.

Group	Parameter	ALFWorld	WebShop
Training	Policy model	Qwen2.5-7B-Instruct	Qwen2.5-7B-Instruct
	RL algorithm	GRPO	GRPO
	Train batch / group size	16 / 16	16 / 8
	Max environment steps	50	15
Retrieval	Skill retrieval mode	embedding	embedding
	General / task skill top- k	3 / 2	3 / 2
Rubric reward	Rubric reward	enabled	enabled
	Judge model	Doubao-1.8	Doubao-1.8
	Max rubric items per skill	2	2
	Reward weights ($\lambda_{\text{env}}, \beta$)	(1.0, 0.2)	(1.0, 0.2)
Skill evolution	Dynamic skill update	enabled	enabled
	Skill update model	Gemini-3.1	Gemini-3.1
	Update interval	5 train steps	10 train steps
	Max new skills per update	2 general + 1 task-specific	2 general + 1 task-specific
	Merge policy	strict, max 1/update	strict, max 1/update
Skill dropout	Skill-on/off rollout mixing	after step 75, 50% off	after step 75, 50% off

Table 6: Key experimental parameters for SkillRubricsRL. We group parameters by training, retrieval, rubric reward, skill-bank evolution, and skill-on/off rollout mixing, and omit infrastructure-only settings such as logging, file paths, and parallelism.

Self-evolved skill	Task-specific principle	Generated rubric criteria
Investigate Cabinets For Soapbars <i>clean</i>	If a soapbar is not immediately visible on a sink or bathtubbasin, systematically open and search bathroom cabinets to locate one. <i>Failure pattern:</i> surface-only bathroom search misses hidden soapbars.	1. Systematic cabinet opening (0.6): opens bathroom cabinets when the soapbar is not found on exposed fixtures. 2. Interior visual inspection (0.4): scans opened cabinet interiors for the soapbar.
Locate Eggs In Fridge <i>heat</i>	If a heating task requires an egg, initiate the search by navigating directly to the fridge, the primary storage location for produce. <i>Failure pattern:</i> generic kitchen search delays checking the fridge.	1. Immediate fridge navigation (0.7): the first navigation action moves directly to the fridge. 2. Fridge interior inspection (0.3): opens the fridge and inspects its contents for the egg.
Target Coffeemachine Destination <i>cool</i>	When placing a cooled object, such as a mug, in a coffeemachine, navigate there directly after retrieving the item from the fridge. <i>Failure pattern:</i> the agent cools the object but wanders before final placement.	1. Immediate coffeemachine navigation (0.6): after retrieving the cooled object, moves directly to the coffeemachine. 2. Final placement execution (0.4): places the cooled object in or on the coffeemachine.
Locate Deskclamp Early <i>look-at-light</i>	Find the furniture hosting the deskclamp early so the agent knows where to illuminate the target object once found. <i>Failure pattern:</i> the target object is found before the lamp host is localized.	1. Early host confirmation (0.6): the deskclamp-hosting furniture enters view within the first 25 actions. 2. Initial navigation priority (0.4): early exploration prioritizes common lamp hosts such as desks or nightstands.
Search Kitchen For Potatoes <i>pick-and-place</i>	When moving a potato, immediately navigate to countertops, cabinets, or the fridge, which are likely produce locations. <i>Failure pattern:</i> broad room exploration delays object acquisition.	1. Initial produce-location navigation (0.5): the first movement goes to a countertop, cabinet, or fridge. 2. Focused receptacle search (0.5): opens and scans likely produce locations while avoiding irrelevant receptacles.

Table 7: Representative self-evolved skill–rubric pairs generated during ALFWorld training. The selected skills are non-duplicative and cover distinct task families: cleaning, heating, cooling, look-at-light, and pick-and-place. Each failure-induced skill is paired with a trajectory-level rubric that turns the discovered behavior into an intrinsic reward signal.

Theme	Generated candidate	Similar existing skill	Merged active skill
initial navigation / anti-stalling	dyn_003: Break Start Paralysis – If unsure how to begin a task, immediately output a 'go to' command targeting the very first receptacle listed in the...	dyn_002: Immediate First Navigation – Upon starting a task, immediately select a receptacle from the initial room observation and use the 'go to' command t...	dyn_005: Immediate First Navigation – Upon starting a task, immediately select a receptacle from the initial room observation and use the 'go to' command to begin exploration and a...
initial navigation / anti-stalling	dyn_007: Initiate Task Immediately – Upon receiving the initial room description at the start of a task, immediately select a receptacle from the observat...	dyn_005: Immediate First Navigation – Upon starting a task, immediately select a receptacle from the initial room observation and use the 'go to' command t...	dyn_009: Immediate First Navigation – Upon starting a task, immediately select a receptacle from the initial room observation and issue a 'go to [receptacle]' command to begin expl...
initial navigation / anti-stalling	dyn_011: Mandatory Initial Navigation – At the start of the task, immediately select a furniture item from the initial room observation and execute a 'go to'...	dyn_009: Immediate First Navigation – Upon starting a task, immediately select a receptacle from the initial room observation and issue a 'go to [receptacl...	dyn_013: Immediate First Navigation – Upon starting a task, immediately select a receptacle or furniture item from the initial room observation and issue a 'go to [receptacle]' com...

Table 8: Examples of semantic skill merging. The updater identifies candidate skills that address an already known failure mode and creates a consolidated replacement while archiving the older version.

Validation evidence	Result (95% CI)
Human agreement on skill applicability	Fleiss' $\kappa = \mathbf{0.601}$ [0.392, 0.782]
Human agreement on rubric criteria	Weighted $\kappa = \mathbf{0.671}$ [0.430, 0.849]
Judge-human agreement on skill execution	Cohen's $\kappa = \mathbf{0.776}$ [0.570, 0.947]
Judge-human skill-execution classification performance	$F_1 = \mathbf{0.928}$ [0.853, 0.984]
Judge-human agreement on rubric scores	Spearman $\rho = \mathbf{0.792}$ [0.617, 0.909]
Judge-human rubric-score error	MAE = $\mathbf{0.127}$ [0.082, 0.175]

Table 9: Human agreement and Judge validity. Confidence intervals are computed with a trajectory-clustered bootstrap over 60 trajectory-skill pairs labeled independently by three expert annotators.

strongly aligned with the human scores (Spearman $\rho = 0.792$; MAE = 0.127). These results support both the operational consistency of the rubric and the validity of the Judge, providing human-grounded evidence for the reliability of Judge-based skill execution rate (SER) estimates.

A.3 Computational Overhead and Approximate Training Cost

The counterfactual utility probes directly reuse the existing GRPO rollout group: one trajectory serves as the full-skill anchor, while each remaining trajectory drops one skill. Consequently, this mechanism neither expands the rollout group nor introduces additional external-model calls. Its overhead is limited to local-policy inference and environment interaction within the existing training procedure.

As shown in Table 10, counterfactual utility probes add no external-model tokens and therefore incur zero additional API cost. Owing to the high cache-hit rate, the effective Doubao-1.8 rubric-Judge cost is approximately \$2.47 for ALFWorld and \$3.02 for WebShop. Gemini-3.1 consumes

Training component	ALFWorld API tokens	WebShop API tokens
Counterfactual utility probes	0	0
Doubao-1.8 rubric Judge	103M	126M
Gemini-3.1 skill evolution	0.80M	0.92M
Total external-model usage	103.80M	126.92M

Table 10: External-model token usage during training. Counterfactual utility probes reuse the existing rollout group and therefore require no additional external-model tokens.

0.80M and 0.92M tokens, respectively, accounting for less than 1% of total external-model token usage in either environment; its API cost is not included because the applicable pricing has not yet been incorporated. Overall, rubric scoring dominates the measured external-model overhead.

Accounting boundary. The token counts above cover external-model usage, whereas the reported dollar amounts cover only Doubao-1.8 rubric-Judge calls. They exclude Gemini-3.1 API charges, actor-training compute, local-policy inference, and

environment rollouts because consistent pricing and local-compute accounting were not available. Table 10 should therefore be interpreted as a measured partial cost rather than the total end-to-end training cost.

A.4 AI Assistance Statement

We used generative AI tools, including ChatGPT, as auxiliary support during manuscript preparation and experimental organization. These tools were employed only to assist the authors in improving efficiency and clarity, without participating in any core research activities such as study design, data collection, or result interpretation. Their use was strictly limited to well-defined, non-critical tasks: language polishing, grammar correction, and clarity editing to refine readability and academic tone; formatting suggestions to align the manuscript with standard scientific writing conventions; code-command drafting for basic script structuring; log summarization to organize experimental records; and consistency checks for reported tables and figures to ensure internal coherence. All AI-generated outputs were reviewed, edited, and verified by the authors, who retain full responsibility for the accuracy and integrity of the work.

A.5 Reproducibility Prompt Templates

For clarity and reproducibility, we collect the main prompt templates used by SkillRubricsRL at the end of the appendix. The templates are organized by their role in the framework: policy interaction and retrieved-skill injection, rubric supervision, and skill-bank evolution and deduplication.

A.5.1 Policy Interaction and Retrieved-Skill Injection

ALFWorld policy prompt with retrieved skills

```
You are an expert agent operating in the ALFRED Embodied
↳ Environment.
Your task is to: {task_description}

## Retrieved Relevant Experience

{retrieved_memories}

## Current Progress

Prior to this step, you have already taken {step_count}
↳ step(s).
Below are the most recent {history_length} observations
↳ and the
corresponding actions you took: {action_history}
You are now at step {current_step} and your current
↳ observation is:
{current_observation}
Your admissible actions of the current situation are:
[{admissible_actions}].

Now it's your turn to take an action.
```

```
Return only one command copied exactly from the admissible
↳ actions list above.
Do not include reasoning, tags, brackets, punctuation, or
↳ extra words.
```

Retrieved skill injection format

```
### General Principles
- **{general_skill_title}**: {general_skill_principle}

### Task-Relevant Skills
- **{task_skill_title}**: {task_skill_principle}
  _Apply when: {task_skill_when_to_apply}_

### Mistakes to Avoid
- **Don't**: {mistake_description}
  **Instead**: {how_to_avoid}
```

WebShop policy prompt with retrieved skills

```
You are an expert autonomous agent operating in the
↳ WebShop e-commerce environment.
Your task is to: {task_description}.

## Retrieved Relevant Experience

{retrieved_memories}

## Current Progress

Prior to this step, you have already taken {step_count}
↳ step(s).
Below are the most recent {history_length} observations
↳ and the
corresponding actions you took: {action_history}
You are now at step {current_step} and your current
↳ observation is:
{current_observation}.
Your admissible actions of the current situation are:
[
{available_actions}
].

Now it's your turn to take one action for the current step.
You should first reason step-by-step about the current
↳ situation, then think
carefully which admissible action best advances the
↳ shopping goal. This
reasoning process MUST be enclosed within <think> </think>
↳ tags.
Once you've finished your reasoning, choose an admissible
↳ action and present it
within <action> </action> tags.
```

A.5.2 Rubric Supervision

Rubric generation prompt

```
You are writing evaluation rubrics for an RL agent skill.
↳ Each rubric item
should be a concrete, observable criterion. Return
↳ 1-{rubric_max_items} rubric
items as JSON array. Each item must have: title,
↳ description, weight. Return
ONLY JSON.

Skill title: {skill_title}
Principle: {skill_principle}
When to apply: {when_to_apply}
```

A.5.3 Skill-Bank Evolution and Deduplication

Failure-to-skill evolution prompt

```
Analyze these failed agent trajectories and suggest NEW
↳ skills to add to the
skill bank.

FAILED TRAJECTORIES:
{failure_examples}
```

TASK TYPES OBSERVED:
{allowed_task_types}

EXISTING SKILL TITLES:
{existing_titles}

Use these to avoid exact duplicates, but do not suppress
→ useful new
specializations merely because they relate to an existing
→ skill.

Generate two kinds of skills:
1. general_skills: cross-task failure fixes that apply
→ across multiple task types.
2. task_specific_skills: failure fixes that apply only to
→ one observed Task Type.

Prioritize reusable failure-mode skills:
- Each skill should target a distinct failure mode, task
→ phase, object
interaction, navigation recovery, or required
→ tool/receptacle behavior.
- Prefer task-specific skills when the failure depends on
→ a concrete task type.
- Do not collapse different task phases into one generic
→ skill.
- Do not generate multiple skills that say the same thing
→ with different wording.
- Abstract across objects and rooms; write skills that
→ would still be useful if
all object names and furniture names changed.

Forbidden overfit skills:
- Do NOT create object-location priors such as "search
→ drawers for watches",
"target sofas for remotes", "locate tomatoes at
→ countertops", or
"check beds for books".
- Do NOT claim an object is usually, commonly, frequently,
→ primarily, or
typically found at a specific furniture/receptacle.
- Do NOT tell the agent to navigate to a specific
→ furniture/receptacle first
solely because of an object type.
- Do NOT encode room-specific assumptions unless the task
→ explicitly requires
that room or tool.
- Mention concrete tools only when they are required by
→ task semantics: fridge
for cool, microwave for heat, sink/faucet for clean,
→ desk lamp for look/examine.

Limits:
- Generate at most {max_new_skills_per_update} general
→ skills.
- For every non-general task category, generate at most
{max_task_specific_skills_per_update} skills.
- task_specific_skills keys must exactly match one of the
→ observed Task Types:
{allowed_task_types}.

Each skill must have: skill_id, title (3-5 words),
→ principle (1-2 sentences),
when_to_apply.
Use skill_ids from this sequence if needed: {example_ids}.
The caller will validate and reassign IDs.

Return ONLY a JSON object, no other text.

condition, same task phase, and same required behavior as
→ the generated skill.
The generated skill may only add wording clarity or
→ missing detail without
changing scope.

Do NOT choose a candidate if the generated skill is a
→ useful specialization, a
complementary behavior, a different failure mode, a
→ different task phase, tied
to different objects/receptacles/tools, or tied to a
→ different task type. If
uncertain, return {"candidate_skill_id": null}.

Return ONLY JSON in this format:
{ "candidate_skill_id": "existing skill_id or null",
"relation": "duplicate|same_scope_detail_update|null",
"confidence": "high|medium|low",
"reason": "short reason" }

Strict skill-merge prompt

Decide whether a generated RL agent skill is the same
→ operational skill as an
existing skill.

Existing skill category: {original_category}
Existing skill:
{original_skill_json}

Generated skill category: {generated_category}
Generated skill:
{generated_skill_json}

Merge only if both skills have the same objective, same
→ trigger condition, same
task phase, and same required behavior. The generated
→ skill may only add clearer
wording or missing detail without changing scope.

Return {"merge": false} if the generated skill is a useful
→ specialization, a
complementary behavior, a different failure mode, a
→ different task phase, tied
to different objects/receptacles/tools, tied to a
→ different task type, or if
both skills could be useful as separate retrieval items.
→ If uncertain, return
{"merge": false}.

Return ONLY JSON with this format:
{
 "merge": true,
 "relation": "duplicate|same_scope_detail_update",
 "confidence": "high",
 "skill": {
 "title": "3-5 words",
 "principle": "1-2 sentences combining both skills",
 "when_to_apply": "clear application condition"
 }
}

Strict merge-candidate selection prompt

Select whether a generated RL agent skill is semantically
→ redundant with an
existing skill.

Generated skill category: {generated_category}
Generated skill:
{generated_skill_json}

Existing candidate skills:
{candidate_skills_json}

Choose one existing skill only when it has the same
→ objective, same trigger