SKILLMASTER: Toward Autonomous Skill Mastery in LLM Agents

Min Yang[1,2], **Jinghua Piao**[2,3] *, **Xu Xia**[2,4], **Yongshun Gong**[1] *, **Yong Li**[2,3] *
[1]Shandong University [2]Zhongguancun Academy [3]Tsinghua University
[4]Southeast University
minyang@mail.sdu.edu.cn Pjh22@mail.tsinghua.edu.cn
s-xx25@bza.edu.cn ysgong@sdu.edu.cn liyong07@tsinghua.edu.cn

Abstract

Skills provide an effective mechanism for improving LLM agents on complex tasks, yet in existing agent frameworks, their creation, refinement, and selection are typically governed by external teachers, hand-designed rules, or auxiliary modules. As a result, skills remain external resources to be invoked, rather than capabilities that agents can develop, adapt, and internalize through experience. To endow LLM agents with autonomous skill mastery, we propose SKILLMASTER, a training framework that teaches agents to create new skills, refine existing skills, and select accumulated skills during task solving. This capability is achieved through three key designs. First, we train agents through trajectory-informed skill review, teaching agents to propose, update, or retain skills based on evidence from completed episodes. Second, each candidate skill edit is designed to be evaluated by its counterfactual utility on related probe tasks, providing a direct learning signal for training skill-editing decisions. Third, we introduce DualAdv-GRPO, which separately estimates advantages for task-solving actions and skill-editing decisions, stabilizing joint training across task solving and skill management. Experiments on ALFWorld, WebShop, and SearchQA show that SKILLMASTER achieves the best overall performance among all compared methods. Further analysis reveals a marked shift in agent capability: agents trained with SKILLMASTER can identify skill failures, refine procedural knowledge from trajectory evidence, and transfer improvements to future tasks with limited skill-bank edits. Overall, SKILLMASTER moves LLM agents beyond mere skill use toward self-improving agents capable of developing, adapting, and applying their own skill repertoires.

1 Introduction

Large language models (LLMs) and LLM-based agents have demonstrated impressive capabilities across complex tasks [Li et al., 2026a] such as few-shot visual recognition [Li et al., 2025a, 2026b], embodied household manipulation [Shridhar et al., 2021, Wang et al., 2023], web navigation [Yao et al., 2022, Liu et al., 2025, Qiu et al., 2025], information seeking [Jin et al., 2025, Zhang et al., 2025, Li et al., 2025b], and software engineering and code generation [Soni et al., 2026, Rashid et al., 2025, Yu et al., 2025a, Puvvadi et al., 2025, Lin et al., 2026]. Despite strong performance on individual tasks, LLM agents remain largely episodic, failing to effectively leverage past experience for cross-task learning [Xia et al., 2026]. Memory-based methods [Chhikara et al., 2025, Liu et al., 2026] can store raw trajectories, but these memories are lengthy and noisy, making it difficult to extract core principles [Xia et al., 2026]. To address this limitation, recent work has introduced the concept

*Jinghua Piao, Yong Li and Yongshun Gong are co-corresponding authors.
[1]Our code is released at `https://github.com/sduyangmin/Skill-Master`.

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

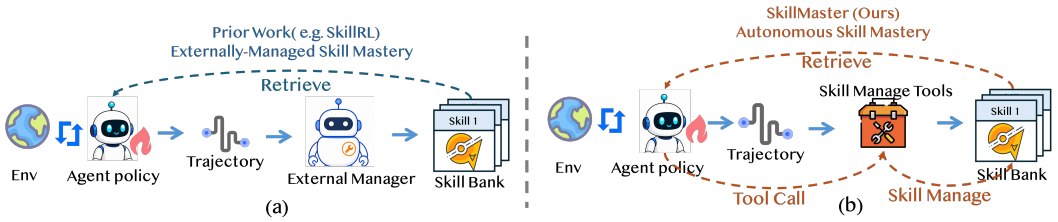


Figure 1: **From externally-managed to autonomous skill mastery. (a) Prior work:** Skill management is handled by an external module; the agent only retrieves skills. **(b) SKILLMASTER:** The agent self-manages its skill bank via tool calls, forming a closed loop where skill management is a learned RL objective.

of skills: compact, reusable abstractions of effective experience that guide future behavior [Xia et al., 2026, Wang et al., 2026, Li et al., 2026c, Jiang et al., 2026, Gao et al., 2026]. Unlike raw trajectories, skills distill essential procedures and heuristics, substantially improving task success and efficiency. The practical value is already evident in deployed systems: personal assistant OpenClaw [OpenClaw Foundation, 2026] and coding agent Claude Code [Anthropic, 2026] both rely on skill-based approaches. However, existing methods [Xia et al., 2026] typically rely on an external LLM teacher that distills skills from completed trajectories on a fixed schedule, while the main policy only retrieves and applies them. Consequently, skill management remains an external mechanism rather than a learnable component of the agent's policy, limiting autonomous skill mastery.

Recent work has attempted to make skill management learnable [Li et al., 2026d, Zhang et al., 2026a, Wu et al., 2025a], but these approaches typically rely on auxiliary modules or separate pipelines. Moreover, skill management is often guided solely by task outcome rewards, which are sparse and coarse, failing to capture how a specific skill edit impacts downstream behavior. This lack of explicit skill-quality signals prevents the agent from fully integrating skill management into its own policy.

Consequently, enabling agents to achieve autonomous skill mastery remains challenging. ***First, skills are externally managed, not internally mastered.*** Skill mastery should be an internalized capability of the agent, rather than an external maintenance procedure. Existing training can teach agents to invoke skills, but it rarely teaches them to treat the skill bank as something they can actively improve from their own experience. ***Second, evaluating skill quality is difficult.*** Task success alone is too sparse to indicate whether a specific skill edit helps. Our key insight is that *high-quality skills should produce two measurable downstream effects: increasing success rates on previously failing tasks and reducing steps on already-solvable tasks.* These observable effects provide the explicit signal missing from external management approaches. ***Third, joint optimization is challenging.*** The optimization objectives for skill management and task execution differ, and combining them in a single policy often causes interference, making training unstable.

To address these challenges, we propose the SKILLMASTER framework, which incorporates skill management into the agent's learning loop. Our framework is built on three key designs. The first, *Trajectory-informed skill review*, allows the agent to use tool-integrated reasoning to propose, update, or retain skills based on completed task trajectories, unifying task execution and skill management in an end-to-end reinforcement learning framework. The second, *Downstream utility reward*, evaluates each candidate skill modification through counterfactual comparisons on related probe tasks, providing an explicit skill-quality signal for training skill-editing decisions. Finally, *DualAdv-GRPO* separately normalizes advantages for task-solving actions and skill-editing decisions, enabling stable joint training of the two optimization objectives within a unified policy.

Our contributions are as follows:

- We propose SKILLMASTER, a framework that integrates task execution with learned skill-management decisions in a single policy, jointly optimized through reinforcement learning.
- We introduce a downstream utility reward that evaluates candidate skill revisions by measuring their counterfactual impact on related probe tasks.
- We propose DualAdv-GRPO, which decouples action optimization from skill management optimization through separate advantage normalization, enabling joint training without objective interference.

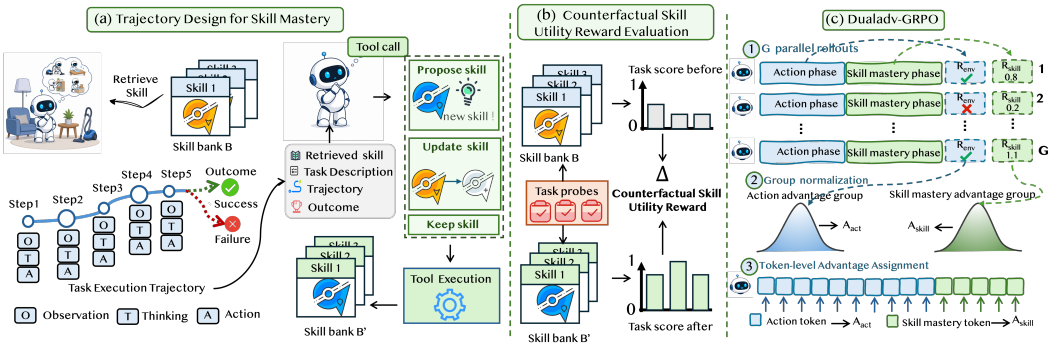


Figure 2: **Overview of SKILLMASTER. (a) Trajectory Design for Skill Mastery:** The agent interacts with the environment guided by retrieved skills and then reviews the episode to propose, update, or retain skills via tool calls. **(b) Counterfactual Skill Utility Reward:** Candidate skill changes are evaluated by counterfactual comparison on related probe tasks. **(c) DualAdv-GRPO:** Action and skill advantages are normalized separately and merged via a tunable weight γ into a unified PPO loss for stable joint training.

- On ALFWorld, WebShop, and SearchQA, SKILLMASTER achieves the best overall performance among all compared methods.

2 Method

Figure 2 provides an overview of the SKILLMASTER framework. We describe its three main components, corresponding to the modules illustrated in the figure. The first component, **Trajectory Design for Skill Mastery** (§2.1), unifies the *Acting Phase* and the *Skill Mastery Phase*: the agent interacts with the environment guided by retrieved skills, and then reviews the episode to propose, update, or retain skills via tool calls. The second component, **Counterfactual Skill Utility Reward** (§2.2), evaluates candidate skill changes by comparing performance on related probe tasks using the original and modified skill banks. The third component, **DualAdv-GRPO** (§2.3) separately normalizes advantages for task-solving actions and skill-editing decisions, merging them via a tunable weight γ into a unified PPO loss to stabilize joint training.

2.1 Trajectory Design for Skill Mastery

SKILLMASTER augments standard agent RL training with an explicit skill-mastery phase appended after each episode. Every episode proceeds in two stages:

- **Acting Phase.** The agent interacts with the environment step by step. At each step, relevant skills are retrieved from the skill bank $\mathcal{B}$ based on the current task and injected into the observation prompt. The agent produces an action and receives a scalar environment reward r_{env}. A trajectory $\tau = \{o_0, a_0, r_0, \ldots, o_T, a_T, r_T\}$ is collected.
- **Skill-Mastery Phase.** After the episode terminates, the system constructs a *skill-review prompt* that presents the task description, the retrieved skills, the trajectory, and the final environment feedback. The agent must then output exactly one tool call: `propose_skill`, `update_skill`, or `keep_skill`. Specially, we expose skill mastery through three function-calling tools: `propose_skill` adds a new skill, `update_skill` revises an existing skill, and `keep_skill` leaves the bank unchanged. Each call is handled by a skill-management backend. During training, candidate modifications generated by `propose_skill` and `update_skill` are temporarily applied to the skill bank and evaluated through counterfactual utility evaluation. A modification is committed to the persistent skill bank only when its utility reward satisfies $R_{\text{utility}} > 0$; otherwise, the modification is discarded. During evaluation, no utility feedback is available, and all valid skill-management tool calls are directly executed and persisted. After each episode, the skill-review prompt presents the task, outcome, retrieved skills, action-observation trajectory, and final environment feedback. The agent is instructed to reason briefly and output exactly one tool call. Full tool

schemas and prompt templates are provided in Appendix E and Appendix A. This phase receives a dedicated skill-mastery reward R_{skill} (defined in §2.2).

The two phases follow distinct optimization objectives: the acting phase is optimized for immediate environment feedback, whereas the skill-mastery phase is optimized for the long-term quality of the skill bank. We therefore treat them as *heterogeneous phases* and introduce a specialized optimization algorithm to train them jointly (§2.3).

2.2 Counterfactual Skill Utility Reward

A central challenge in self-managed skill evolution is defining what makes a skill *good*. In prior work such as SKILLRL [Xia et al., 2026], skills are distilled by an external teacher and the agent is trained with only the environment outcome reward—a binary success signal that conflates task execution quality with skill quality. This reward cannot distinguish a genuinely useful skill from one that merely *looks* plausible, because it provides no signal about whether the skill actually helps on future tasks.

Our key insight is that a high-quality skill should produce two measurable effects on tasks that share similar requirements: (1) tasks that previously *failed* should become more likely to *succeed*, and (2) tasks that already succeeded should be completable in *fewer steps*, since the skill encodes more efficient strategies. We operationalize this intuition through a *downstream utility reward* based on counterfactual probe evaluation.

2.2.1 Probe-Based Counterfactual Evaluation

When the agent calls `propose_skill` or `update_skill`, we select K *probe tasks* that are semantically related to the current episode (e.g., tasks from the same task family, which share similar skill requirements). The selection uses a deterministic seed derived from the current task identifier, ensuring reproducibility. The probe pool is drawn from a held-out set of tasks from the training, so utility evaluation measures genuine skill transfer rather than memorization of seen tasks. The benchmark-specific definitions of probe task are detailed in §3.1.

For each probe task p_i, we evaluate the impact of a candidate skill modification by comparing the task under two skill banks:

$$\mathcal{B} \xrightarrow{\text{apply candidate mutation}} \mathcal{B}'.$$

Specifically, we first rollout p_i using the original skill bank $\mathcal{B}$. Next, we create a temporary bank $\mathcal{B}'$ by applying the candidate skill modification, and then rollout the same probe task p_i using $\mathcal{B}'$. This counterfactual comparison provides a direct measure of the skill modification's effect on task performance.

Each probe rollout is scored according to our two desiderata—success and efficiency:

$$\text{score}(p_i, \mathcal{B}) = \mathbf{1}[\text{success}_i] \; + \; \frac{M - \text{steps}_i}{M} \cdot \mathbf{1}[\text{success}_i] \tag{1}$$

where M is the maximum allowed steps. The first term captures whether the task now succeeds where it may have failed before; the second term captures step efficiency—a skill that encodes a shorter, more direct strategy yields a higher score. Failed probes receive 0, since a skill that does not enable success confers no benefit regardless of step count.

2.2.2 Utility Computation

Define $\delta_i = \text{score}(p_i, \mathcal{B}') - \text{score}(p_i, \mathcal{B})$ as the per-probe performance delta. Over K probes, let:

$$\bar{\delta} = \frac{1}{K} \sum_{i=1}^{K} \delta_i, \qquad w = \sum_{i=1}^{K} \mathbf{1}[\delta_i > 0], \qquad \ell = \sum_{i=1}^{K} \mathbf{1}[\delta_i < 0] \tag{2}$$

The utility reward combines the average improvement magnitude with a directional consistency term:

$$R_{\text{utility}} = \bar{\delta} + \alpha \cdot \frac{w - \ell}{K}, \tag{3}$$

here, α balances the magnitude of improvement and directional consistency. The directional consistency term $\alpha \cdot (w - \ell)/K$ reduces sensitivity to single-probe outliers, favoring skill edits whose effects are supported by multiple probes. This design encourages skill exploration while ensuring that rewards are assigned only to edits with consistent downstream impact. The full skill-mastery reward is:

$$R_{\text{skill}} = R_{\text{format}} + R_{\text{utility}} \tag{4}$$

where R_{format} is a composite correctness reward that gives a small positive bonus for valid tool execution and penalizes parse errors, missing `<think>` or `<tool_call>` tags, and placeholder content.

2.3 DualAdv-GRPO: Decoupled Optimization over Heterogeneous Phases

Standard GRPO normalizes rewards within a group of G rollouts, implicitly assuming that rewards share a common scale and semantics. This assumption fails for SKILLMASTER: acting phases receive binary task rewards r_{env}, whereas skill-mastery phases receive continuous rewards R_{skill} from tool validity and probe utility. Normalizing them together would entangle task execution with skill evolution and distort within-type preference ordering.

2.3.1 Dual-Stream Advantage Estimation

We address this through *dual-stream advantage normalization*. For each prompt, we sample G independent trajectories, each containing acting phases followed by one skill-mastery phase. We compute separate GRPO statistics for the two reward streams: $(\mu_{\text{act}}, \sigma_{\text{act}})$ from $\{r_{\text{env}}^{(j)}\}_{j=1}^{G}$ and $(\mu_{\text{skill}}, \sigma_{\text{skill}})$ from $\{R_{\text{skill}}^{(j)}\}_{j=1}^{G}$. The corresponding normalized advantages are:

$$A_{\text{act}}^{(j)} = \frac{r_{\text{env}}^{(j)} - \mu_{\text{act}}}{\sigma_{\text{act}} + \epsilon}, \qquad A_{\text{skill}}^{(j)} = \frac{R_{\text{skill}}^{(j)} - \mu_{\text{skill}}}{\sigma_{\text{skill}} + \epsilon}, \tag{5}$$

where ϵ is a small constant for numerical stability. Thus, advantages assigned to action tokens are normalized only against action rewards from other trajectories of the same prompt, while advantages assigned to skill-mastery tokens are normalized only against skill-mastery rewards from the same prompt group.

2.3.2 Type-Conditioned Policy Gradient

We next merge the two normalized advantage streams at the token level. For each trajectory j, let $S_{\text{act}}^{(j)}$ and $S_{\text{skill}}^{(j)}$ denote the token positions belonging to acting phases and the skill-mastery phase, respectively. DualAdv-GRPO assigns advantages according to token type:

$$A_{\text{DualAdv}}^{(j)}(l) = \begin{cases} A_{\text{act}}^{(j)}, & l \in S_{\text{act}}^{(j)} \\ \gamma \cdot A_{\text{skill}}^{(j)}, & l \in S_{\text{skill}}^{(j)} \end{cases} \tag{6}$$

where $\gamma > 0$ controls the relative weight of the skill-mastery objective. This preserves within-type preference ordering while allowing action learning and skill-mastery learning to update a single policy.

The per-token policy gradient is computed through the standard PPO-clipped objective. Let $r_l(\theta) = \pi_\theta(a_l \mid o_l)/\pi_{\text{old}}(a_l \mid o_l)$ be the probability ratio and L the number of generated tokens:

$$\mathcal{L}^{(j)}(\theta) = -\frac{1}{L}\sum_{l=1}^{L} \min\Big(r_l(\theta)\, A_{\text{DualAdv}}^{(j)}(l),\ \text{clip}\big(r_l(\theta),\, 1-\varepsilon,\, 1+\varepsilon\big)\, A_{\text{DualAdv}}^{(j)}(l)\Big) \tag{7}$$

The full objective averages over all N trajectories with KL regularization:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{j=1}^{N} \mathcal{L}^{(j)}(\theta) \ + \ \beta \cdot \mathcal{L}_{\text{KL}}(\theta) \tag{8}$$

Table 1: **Performance comparison on ALFWorld and WebShop,** where we report ALFWorld per-family success rates (%) and overall average success rate, together with WebShop score and success rate. The best results are shown in **bold** and the second-best results are underlined.

	ALFWorld							WebShop	
Method	**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
Prompt-based Agentic or Memory-based Methods									
Qwen2.5-7B-Instruct	33.4	21.6	19.3	6.90	2.80	3.20	14.8	26.4	7.80
ReAct	48.5	35.4	34.3	13.2	18.2	17.6	31.2	46.2	19.5
Reflexion	62.0	41.6	44.9	30.9	36.3	23.8	42.7	58.1	28.8
Mem0	54.0	55.0	26.9	36.4	20.8	7.69	33.6	23.9	2.00
ExpeL	21.0	67.0	55.0	52.0	71.0	6.00	46.3	30.9	11.2
MemP	54.3	38.5	48.1	56.2	32.0	16.7	41.4	25.3	6.40
SimpleMem	64.5	33.3	20.0	12.5	33.3	3.84	29.7	33.2	8.59
RL-based Methods									
RLOO	87.6	78.2	87.3	81.3	71.9	48.9	75.5	80.3	65.7
GRPO	90.8	66.1	89.3	74.7	72.5	64.7	77.6	79.3	66.1
Memory-Augmented RL-based Methods									
MemRL	62.8	38.5	22.2	12.5	8.00	0.00	21.4	29.5	9.20
EvolveR	64.9	33.3	46.4	13.3	33.3	33.3	43.8	42.5	17.6
Mem0+GRPO	78.1	54.8	56.1	31.0	65.0	26.9	54.7	58.1	37.5
SimpleMem+GRPO	89.5	36.3	60.0	50.0	64.9	26.3	62.5	67.8	46.9
SkillRL	97.9	71.4	90.0	90.0	95.5	87.5	89.9	85.2	72.7
SkillMaster	**100**	**100**	**100**	**97.1**	**95.7**	**100**	**98.7**	**95.0**	**82.0**

The complete DualAdv-GRPO procedure is summarized in Algorithm 1 in Appendix B.

3 Experiments

We evaluate SkillMaster on three standard LLM-agent benchmarks: ALFWorld, WebShop and SearchQA. Our experiments are organized around four research questions:

Q1. Does SkillMaster outperform state-of-the-art methods including closed-source LLMs, prompt-based agents, and RL-based approaches?

Q2. How do individual components affect performance?

Q3. How does SkillMaster evolve and manage the skill bank?

Q4. How does SkillMaster manage skills in practice?

3.1 Experimental Setup

Environments. We evaluate on ALFWorld [Shridhar et al., 2021], an embodied household benchmark spanning six manipulation task families, and WebShop [Yao et al., 2022], an online shopping benchmark requiring multi-step product search and purchase. We also evaluate on SearchQA, which covers single-hop and multi-hop question answering across seven datasets. Detailed task descriptions are provided in Appendix G.

Baselines. We compare against four categories of methods: closed-source LLMs as zero-shot agents (GPT-4o, Gemini-2.5-Pro); prompt-based and memory-augmented frameworks (ReAct, Reflexion, Mem0, ExpeL, MemP); standard RL methods (RLOO, GRPO); memory-augmented RL approaches (EvolveR, MemRL, Mem0+GRPO, SimpleMem+GRPO); and SKILLRL as the state-of-the-art teacher-driven skill evolution baseline. Full baseline descriptions are provided in Appendix D. A detailed fair comparison statement is provided in Appendix J.

Table 2: Performance comparison on search-augmented QA tasks (%). Baseline results are taken from Xia et al. [2026]. The best results are shown in **bold**, and the second-best results are underlined.

Method	Single-Hop QA			Multi-Hop QA				Overall
	NQ	TriviaQA	PopQA	HotpotQA	2Wiki	MuSiQue	Bamboogle	
Qwen2.5	11.6	35.6	1.20	16.4	22.2	4.80	14.4	15.2
CoT	12.8	35.6	3.80	16.2	22.6	6.60	24.0	17.4
RAG	27.4	58.2	17.8	25.8	23.2	9.40	16.8	25.5
Search-o1	19.4	40.6	11.4	17.0	27.0	8.60	30.4	22.1
R1-Instruct	21.0	44.9	17.1	20.8	27.5	6.00	19.2	22.4
Search-R1	39.3	61.0	39.7	37.0	40.1	14.6	36.8	38.5
ZeroSearch	43.6	61.8	**51.5**	34.6	35.2	18.4	27.8	39.1
StepSearch	–	–	–	38.6	36.6	**22.6**	40.0	–
EvolveR	43.5	**63.4**	44.6	38.2	**42.0**	15.6	54.4	43.1
SKILLRL	45.9	63.3	45.9	43.2	40.3	20.2	**73.8**	47.1
SKILLMASTER	**49.1**	62.9	47.0	**45.6**	41.6	20.5	70.7	**48.0**

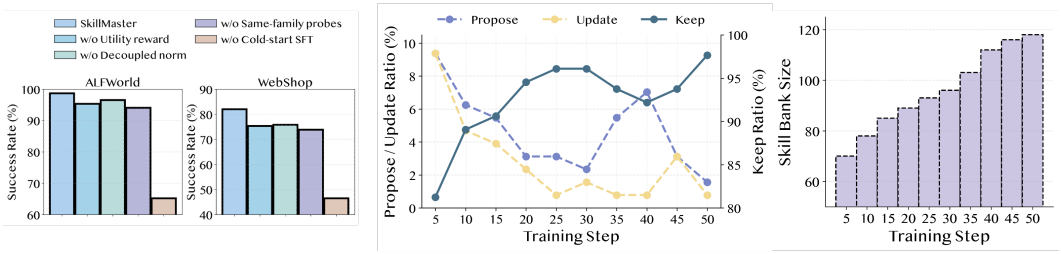
Implementation Details. We follow the cold-start SFT pipeline of SKILLRL [Xia et al., 2026], using Claude as a teacher to generate skill-augmented reasoning traces for the Qwen2.5-7B-Instruct base model. RL training uses GRPO with group size $G = 8$, KL penalty coefficient 0.01, and learning rate 1×10^{-6}, on 8×A100 GPUs via the Verl framework with vLLM for rollout generation. The main results of SKILLMASTER are averaged over three runs. All baseline results are taken from SKILLRL [Xia et al., 2026]. The initial skill bank is adapted from SKILLRL with light deduplication. Skill-mastery phases are appended after every training episode. The utility reward uses $K = 4$ probes with $\alpha = 0.3$, selected by task family for ALFWorld, by product category for WebShop and by semantic similarity between training questions for SearchQA. Full configuration details are provided in Appendix C.

3.2 Main Results

Table 1 confirms two baseline trends. First, RL-trained agents substantially outperform closed-source LLMs and prompt-based methods, confirming that policy optimization is critical for interactive tasks. Second, structured skill libraries (SkillRL) significantly outperform generic memory augmentation, demonstrating the value of procedural abstraction over raw trajectory storage.

SKILLMASTER achieves the best performance while removing the external teacher. On ALFWorld, it raises the strongest baseline from 89.9% to 98.7%, with perfect success on four of six task families and above 95% on the remaining two. The average improvement of 8.8 percentage points over SkillRL is particularly notable given that SkillRL already represents a strong starting point near 90% success. On WebShop, success improves from 72.7% to 82.0%, and the overall score increases from 85.2 to 95.0. The gap between score (95.0) and success (82.0) suggests that SKILLMASTER not only completes more tasks but also achieves higher-quality decisions on tasks it successfully solves. These gains are broadly distributed across all task types, with no family falling below 95% on ALFWorld, indicating improved general competence rather than overfitting to specific patterns. Table 2 further reports performance on seven search-augmented QA datasets covering single-hop and multi-hop questions. SKILLMASTER achieves the highest overall score of 48.0%, exceeding SKILLRL by 0.9 percentage points. It improves over SKILLRL on five of the seven datasets and achieves the best performance on NQ and HotpotQA. Means and standard deviations across repeated runs on ALFWorld, WebShop, and SearchQA are reported in Appendix O.

Unlike the teacher-driven approach in SkillRL, SKILLMASTER learns skill management end-to-end as part of its policy. The downstream utility reward provides a direct quality signal for skill edits via counterfactual comparison on probe tasks, and the agent learns to decide *when* and *how* to revise skills from trajectory evidence. The substantial gains over SkillRL in Table 1 demonstrate the advantage of this utility-guided design over fixed external curation. A detailed case study appears in Section 3.5.



(a) (b) (c)

Figure 3: **(a)** Ablation of skill mastery components on ALFWorld and WebShop. **(b)** Evolution of skill-management behaviors during ALFWorld RL training, showing the changing proportions of `keep_skill`, `propose_skill`, and `update_skill`. **(c)** Growth of the skill bank size during ALFWorld RL training.

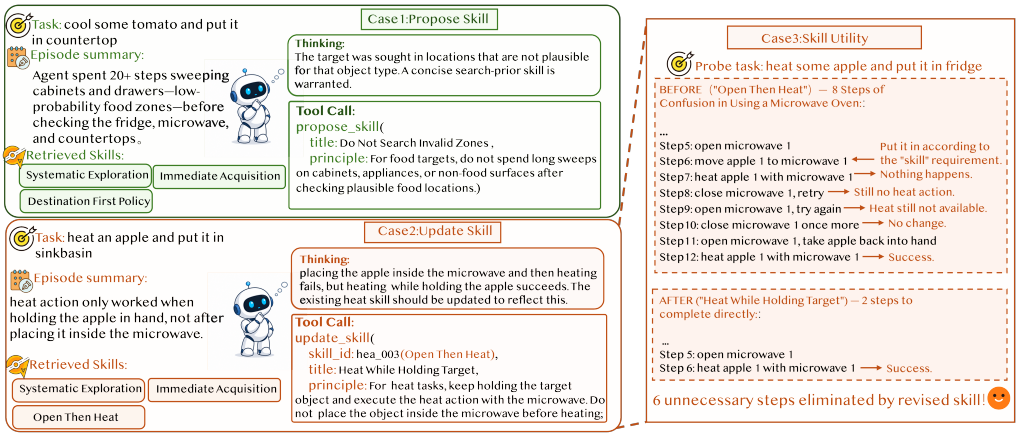


Figure 4: **Case studies of skill management and utility evaluation. Case 1 (Propose Skill):** The agent failed a cooling task after exhaustively searching low-probability food zones (cabinets, drawers). It proposed `Do Not Search Invalid Zones`, encoding a reusable search-prior principle. **Case 2 (Update Skill):** The agent observed that heating succeeded only when holding the apple, contradicting the existing skill `Open Then Heat` (which instructed placing the object inside first). It revised the skill to `Heat While Holding Target`. **Case 3 (Skill Utility):** A probe task (`heat apple` → `fridge`) was evaluated before and after the skill revision of Case 2. Before: the old skill caused an 8-step microwave confusion (place inside → heat fails → repeated open/close → take back → re-heat). After: the revised skill enabled correct 2-step execution (open microwave, heat while holding), eliminating 6 unnecessary steps.

3.3 Ablation Study

Figure 3(a) reports the impact of removing each component of SKILLMASTER. Removing the utility reward leads to a clear drop on both benchmarks, suggesting that counterfactual probe evaluation provides an important signal for distinguishing useful skill edits from merely well-formed ones. Replacing per-type advantage normalization with a single coupled normalization group also reduces performance, showing that decoupling action and skill-mastery advantages is important for joint optimization. Random probes consistently underperform family probes, showing that task-related probe selection is important for informative skill-quality feedback. Removing the cold-start SFT phase causes the largest degradation, highlighting the importance of initializing the agent with basic tool-use and task-execution capabilities. Overall, each component contributes to stable and effective self-managed skill evolution. To further disentangle the effect of utility-credited skill evolution from extra post-episode computation, and to test robustness to weaker initial skill banks, we provide additional control experiments in Appendix L and Appendix M.

3.4 Analysis of Skill Evolution

We further analyze the skill evolution process on ALFWorld during the RL training stage, where the agent performs skill-management operations after each episode. Beyond downstream task performance, we examine whether SKILLMASTER can autonomously create, refine, and retain skills as the skill bank evolves. Figure 3(b) shows the evolution of skill-management decisions, including `keep_skill`, `propose_skill`, and `update_skill`, during ALFWorld training. Early in training, the agent performs more active skill modifications, with `propose_skill` and `update_skill` accounting for 18.76% of decisions. As training progresses, the agent gradually shifts toward skill retention, with `keep_skill` increasing from 81.25% to 97.66% while modification decisions decrease to 2.34%. This transition indicates that SKILLMASTER learns to explore new skills initially and then consolidate useful knowledge as the skill bank matures. Figure 3(c) further shows the growth of the skill bank during ALFWorld training. Starting from 58 initial skills, the bank expands to 118 skills after training, demonstrating that the learned skill-management policy can continuously discover and preserve new procedural knowledge. Frozen-checkpoint evaluations further show that SKILLMASTER maintains strong performance after restoring the original initial bank and can construct useful skills from an empty bank at test time. Detailed results are provided in Appendix N.

3.5 Case Study

Figure 4 presents three cases drawn from ALFWorld rollouts, illustrating how SkillMaster manages its skill bank through TIR. Full case descriptions are provided in Appendix H.

Case 1 demonstrates *proposing* a new skill when a gap is detected. The agent failed a cooling task because it swept low-probability food zones instead of checking high-probability locations first. During skill review, it recognized that none of the retrieved skills addressed search-prior allocation, and called `propose_skill` to add a reusable search heuristic.

Case 2 demonstrates *updating* an existing skill when it proves imprecise. The agent succeeded at a heating task but observed a recurring pattern: the existing skill instructed placing the object inside the microwave before heating, yet heating only worked when the object was held in hand. The agent called `update_skill` to correct this operational guidance.

Case 3 *validates* the downstream impact of the revision from Case 2. A probe task from the same family was evaluated before and after the skill change. Before the revision, the imprecise skill caused an 8-step confusion at the microwave. After the revision, the same task completed heating in 2 steps, eliminating six unnecessary operations. This direct efficiency gain is precisely what the utility reward is designed to measure and reinforce.

3.6 Computational Overhead

We also analyze computational overhead in Appendix K. On ALFWorld, enabling utility-based skill management increases the average training time from 12.13 minutes per step to about 16.00 minutes per step, corresponding to a 31.9% wall-clock increase. It also raises average GPU memory allocation from 33.51 GiB to 34.57 GiB and peak GPU memory allocation from 61.29 GiB to 70.49 GiB. These results show that the utility reward introduces noticeable but manageable computational overhead.

4 Related Work

We summarize the most relevant work here and defer a broader discussion to Appendix I.

Skill management in LLM agents. Prior agent-memory methods store interaction traces for later retrieval [Park et al., 2023, Shinn et al., 2023, Chhikara et al., 2025], whereas skill-based methods abstract experience into reusable procedural knowledge [Wang et al., 2023, Xia et al., 2026]. In most such approaches, however, skill management is handled by an external teacher, fixed update rule, or specialized module rather than by the acting policy itself. Recent concurrent methods make skill evolution more adaptive but still rely on managerial roles, controllers, or skill-generation pipelines [Li et al., 2026d, Zhang et al., 2026a, Wu et al., 2025a, Wang et al., 2025, Zhang et al., 2026b]. Moreover, they typically assess skill quality only indirectly through task outcomes or heuristic update rules, rather than explicitly measuring the causal utility of a particular skill edit. It also leaves skill editing

weakly supervised, since task-level rewards need not reveal whether a specific write operation was beneficial. SKILLMASTER instead uses tool-integrated reasoning to let the same policy both act and edit its own skill bank, with edits credited by counterfactual downstream utility.

RL and tool-integrated reasoning. RL post-training methods such as PPO, RLOO, GRPO, DAPO, and GiGPO optimize LLM agents through environment feedback [Schulman et al., 2017, Ahmadian et al., 2024, Shao et al., 2024, Yu et al., 2025b, Feng et al., 2025]. Tool-use and TIR methods train models to invoke tools or structured function calls [Schick et al., 2023, Wei et al., 2025, Fang and Sun, 2026, Zeng et al., 2026, Zhang et al., 2026c]. Most such methods use tools to access external information or execute auxiliary computation, rather than to modify the agent's own reusable memory. Our setting is different because memory-write actions affect future behavior only indirectly, often across multiple later episodes. As a result, action decisions and write decisions induce different credit-assignment structures and should not share a single normalized advantage signal. Our work extends this paradigm to self-editing skill memory, where write actions are evaluated by downstream utility and optimized separately from task-execution turns.

5 Conclusion and Discussion

We presented SKILLMASTER, a framework that enables LLM agents to manage reusable skill banks through tool-integrated reasoning and reinforcement learning. The central idea is to turn skill management from an externally managed maintenance process into an optimization problem within the agent's own policy, where skill changes are generated through tool calls and credited by counterfactual downstream utility. This enables the policy to jointly learn how to act and how to improve the knowledge it will reuse in future episodes. Experiments on ALFWorld, WebShop, and SearchQA show that SKILLMASTER outperforms the compared baselines overall. Ablations on ALFWorld and WebShop further support the contributions of the utility reward and decoupled optimization. These results suggest that learned skill management can be a useful mechanism for improving the quality of reusable procedural knowledge in LLM agents. At the same time, the current method relies on predefined task groupings or semantic similarity for probe selection, and the cost of probe evaluation grows with episode length and mutation frequency, even if the overhead is manageable in our current settings. Moreover, our evaluation is limited to structured interactive environments and search-augmented QA, and further validation in more open-ended environments is needed. Promising directions for future work include skill deletion, multi-agent skill sharing, continual learning under distribution shift, and extending TIR to other forms of agent self-improvement. A more detailed discussion is provided in Appendix F.

Acknowledgments and Disclosure of Funding

This work was supported by the National Key Research and Development Program of China (Grant No. 2024YFC3307603), the National Natural Science Foundation of China (Grant No. U23B2030), and Zhongguancun Academy (Grant No. C20250201).

References

Wenhao Li, Zhibin Wu, Chong Xiao, and Qiangchang Wang. Semmsa: Latent semantic-aided robust multimodal sentiment analysis with incomplete data. *arXiv preprint arXiv:2609.30238*, 2026a.

Wenhao Li, Qiangchang Wang, Xianjing Meng, Zhibin Wu, and Yilong Yin. Vt-fsl: Bridging vision and text with llms for few-shot learning. In *NeurIPS*, volume 38, pages 113697–113727, 2025a.

Wenhao Li, Xianjing Meng, Qiangchang Wang, Zhongyi Han, Zhibin Wu, and Yilong Yin. Dvla-rl: Dual-level vision-language alignment with reinforcement learning gating for few-shot learning. In *ICLR*, 2026b.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

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.

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.

Jiarun Liu, Jia Hao, Chunhong Zhang, and Zheng Hu. Wepo: Web element preference optimization for llm-based web navigation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 26614–26622, 2025.

Haoyi Qiu, Alexander Richard Fabbri, Divyansh Agarwal, Kung-Hsiang Huang, Sarah Tan, Nanyun Peng, and Chien-Sheng Wu. Evaluating cultural and social awareness of llm web agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3978–4005, 2025.

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.

Ding-Chu Zhang, Yida Zhao, Jialong Wu, Liwen Zhang, Baixuan Li, Wenbiao Yin, Yong Jiang, Yu-Feng Li, Kewei Tu, Pengjun Xie, et al. Evolvesearch: An iterative self-evolving search agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13134–13147, 2025.

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, 2025b.

Aditya Bharat Soni, Boxuan Li, Xingyao Wang, Valerie Chen, and Graham Neubig. Coding agents with multimodal browsing are generalist problem solvers. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 6052–6069, 2026.

Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, et al. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703*, 2025.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. Utboost: Rigorous evaluation of coding agents on swe-bench. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3762–3774, 2025a.

Meghana Puvvadi, Sai Kumar Arava, Adarsh Santoria, Sesha Sai Prasanna Chennupati, and Harsha Vardhan Puvvadi. Coding agents: A comprehensive survey of automated bug fixing systems and benchmarks. In *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 680–686. IEEE, 2025.

Yexiong Lin, Shanshan Ye, Yu Yao, Zhen Fang, Bo Han, and Tongliang Liu. Squidagent: Parallelize wisely, coordinate efficiently. *arXiv preprint arXiv:2610.08647*, 2026.

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.

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.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026.

Junda Wang, Zonghai Tao, Hansi Zeng, Zhichao Yang, Hamed Zamani, and Hong Yu. Tarse: Test-time adaptation via retrieval of skills and experience for reasoning agents. *arXiv preprint arXiv:2603.01241*, 2026.

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. *arXiv preprint arXiv:2602.12670*, 2026c.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills–beyond tool use in llm agents. *arXiv preprint arXiv:2602.20867*, 2026.

Yudong Gao, Zongjie Li, Zimo Ji, Pingchuan Ma, Shuai Wang, et al. Skillreducer: Optimizing llm agent skills for token efficiency. *arXiv preprint arXiv:2603.29919*, 2026.

OpenClaw Foundation. Openclaw documentation. `https://docs.openclaw.ai/index`, 2026. Accessed: 2026-05-07.

Anthropic. Claude code. `https://www.anthropic.com/product/claude-code`, 2026. Accessed: 2026-05-07.

Yu Li, Rui Miao, Zhengling Qi, and Tian Lan. Arise: Agent reasoning with intrinsic skill evolution in hierarchical reinforcement learning. *arXiv preprint arXiv:2603.16060*, 2026d.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. Co-evolving llm decision and skill bank agents for long-horizon tasks. *arXiv preprint arXiv:2604.20987*, 2025a.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

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.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, et al. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, 2024.

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.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025b.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Yifan Wei, Xiaoyan Yu, Yixuan Weng, Tengfei Pan, Angsheng Li, and Li Du. Autotir: Autonomous tools integrated reasoning via reinforcement learning. *arXiv preprint arXiv:2507.21836*, 2025.

Zhaiyu Fang and Ruipeng Sun. Adatir: Adaptive tool-integrated reasoning via difficulty-aware policy optimization. *arXiv preprint arXiv:2601.14696*, 2026.

Yirong Zeng, Xiao Ding, Yufei Liu, Yuxian Wang, Qunyao Du, Yutai Hou, Wu Ning, Haonan Song, Duyu Tang, Dandan Tu, et al. Autotool: Automatic scaling of tool-use capabilities in rl via decoupled entropy constraints. *arXiv preprint arXiv:2603.13348*, 2026.

Xuqin Zhang, Quan He, Zhenrui Zheng, Zongzhang Zhang, Xu He, and Dong Li. Aster: Agentic scaling with tool-integrated extended reasoning. *arXiv preprint arXiv:2602.01204*, 2026c.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026d.

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 preprint arXiv:2508.06433*, 2025.

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*, 2025b.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565, 2024.

A Skill-Review Prompt Templates

Figures 5 and 6 show the full skill-review prompt templates used in SKILLMASTER for ALFWorld and WebShop, respectively. The prompts are constructed by `build_skill_management_prompt` in `skill_management.py`. Placeholders (`{task}`, `{category}`, etc.) are filled per-episode with the task description, inferred skill category, retrieved skills, trajectory trace, and outcome.

B DualAdv-GRPO Algorithm

Algorithm 1 summarizes one training iteration of the DualAdv-GRPO optimization loop.

Algorithm 1 DualAdv-GRPO: One Training Iteration

Require: Policy π_θ, reference policy π_{ref}, skill bank $\mathcal{B}$, environment $\mathcal{E}$, group size G, skill weight γ

for each prompt p in batch **do**
  for $g = 1$ to G **do**
    Sample trajectory τ_g from π_θ: act in $\mathcal{E}$ with skills from $\mathcal{B}$, then output one skill-management tool call
    Collect $r_{\text{env}}^{(g)}$ and $R_{\text{skill}}^{(g)}$
  end for
  Compute $A_{\text{act}}^{(g)}$, $A_{\text{skill}}^{(g)}$ via Eq. 5 using per-type group statistics
  for each token l of τ_g **do**
    if $l \in S_{\text{act}}^{(g)}$ **then**
      Use advantage $A_{\text{act}}^{(g)}$
    else
      Use advantage $A_{\text{skill}}^{(g)}$ weighted by γ
    end if
  end for
end for
$\mathcal{L} \leftarrow \frac{1}{N} \sum_{j=1}^{N}$ Eq. 7 $+ \beta \cdot \mathcal{L}_{\text{KL}}$
$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
return Updated policy π_θ

C Implementation and Configuration Details

Probe task selection. For ALFWorld, probe tasks are selected from the same task family as the current episode, which forms a natural skill-sharing boundary: a skill about microwave usage should affect all heat tasks but is unlikely to affect clean or pick-and-place tasks. For WebShop, probes are drawn from the same product category (apparel, footwear, electronics, accessories, home decor, beauty, health), under the heuristic that skills for searching and filtering shirts transfer to other apparel tasks while being largely irrelevant to electronics or footwear. The selection uses a deterministic seed derived from the current task identifier to ensure reproducibility while avoiding overlap with the current task. Probe rollouts use a separate environment pool with a distinct seed offset by 50,000 to avoid data leakage. SearchQA does not provide predefined task families. We therefore construct an offline semantic probe index over the training questions using Qwen3-Embedding-0.6B. For each training question, we select the K most similar questions according to cosine similarity, excluding the question itself, and store their identifiers in advance. During RL training, probes are retrieved through direct table lookup, without online embedding computation or search. Across all three benchmarks, probe candidates are drawn exclusively from the training split; validation and test examples are never used for probe selection or utility evaluation.

Training hyperparameters. All experiments use GRPO with group size $G = 8$, a KL penalty coefficient of 0.01, and a constant learning rate of 1×10^{-6}. The utility reward uses $K = 4$ probes and $\alpha = 0.3$. Skill-management turns are configured at trajectory level. The skill bank uses the standard SKILLRL JSON format with general skills and task-specific skills organized by category.

D Baseline Descriptions

Our baselines span four categories. First, we include strong closed-source LLMs as zero-shot agents: GPT-4o and Gemini-2.5-Pro serve as upper bounds for general-purpose reasoning without task-specific training. Second, we compare against prompt-based and memory-augmented agent frameworks: ReAct and Reflexion use in-context reasoning chains to improve multi-step decision making; Mem0, ExpeL, and MemP maintain external memory stores or experience pools to guide agent behavior, but do not update model parameters through reinforcement learning. Third, we evaluate standard RL-based methods including RLOO and GRPO, which optimize agent policies through group-relative advantage estimation over trajectory rollouts. Fourth, we consider memory-augmented RL approaches that embed persistent memory mechanisms directly into the RL optimization loop: EvolveR, MemRL, Mem0 combined with GRPO, and SimpleMem combined with GRPO. Finally, we include SKILLRL as a direct point of comparison, since it represents the state of the art in teacher-driven skill evolution and shares the same skill bank infrastructure.

E Tool Schemas

The three skill-management tools are registered in the multi-turn rollout infrastructure as standard OpenAI-compatible function-calling schemas. Figures 7, 8, and 9 show the exact JSON definitions passed to the LLM during the skill-management turn.

F Extended Discussion

Probe selection. The current probe selection uses task families for ALFWorld, product categories for WebShop, and semantic similarity between training questions for SearchQA. In more open-ended environments, task similarity may not reliably indicate whether a skill will transfer. Improving probe selection in such settings remains a direction for future work.

Probe evaluation cost. Each skill mutation triggers O(K) additional probe rollouts. While the overhead is modest in our settings, it would grow for environments with longer episodes or for skill banks that trigger frequent mutations. Strategies such as importance sampling over probes, amortized evaluation across multiple candidate mutations, or learned value functions that approximate probe scores could reduce this cost in future work.

Domain scope. Our evaluation covers structured interactive environments and search-augmented QA. Whether SkillMaster transfers to more open-ended environments, such as software engineering, remains to be validated. The core mechanism is domain-agnostic: any environment that admits a notion of related tasks can in principle support counterfactual probe evaluation.

Future directions. Beyond addressing the limitations above, several research directions are promising. First, the utility reward framework could support skill deletion, enabling the agent to identify and remove outdated or harmful skills rather than only adding and revising them. Second, multi-agent skill sharing, where agents trained on different task distributions exchange skills through a shared bank, could accelerate collective learning. Third, combining SkillMaster with continual learning setups where the task distribution shifts over time would test whether self-managed skill evolution enables faster adaptation compared to static baselines. Finally, extending TIR to other forms of agent self-improvement, such as prompt refinement or tool composition, could broaden the impact of this approach.

G Environment Details

ALFWorld is an embodied household benchmark where the agent navigates and manipulates objects in simulated indoor scenes to complete goal-directed tasks. It contains six task families: Pick and Place requires retrieving and delivering objects to specified locations; Look at Object involves examining items under a desklamp; Clean requires washing objects at a sink; Heat requires warming objects using a microwave; Cool requires chilling objects in a fridge; and Pick Two requires collecting and delivering two objects. WebShop is an online shopping benchmark where the agent must search

for products, navigate result pages, select appropriate options such as size and color, and complete purchases to match a given goal specification. SearchQA evaluates search-augmented question answering on seven datasets: NQ, TriviaQA, PopQA, HotpotQA, 2Wiki, MuSiQue, and Bamboogle. These datasets cover both single-hop and multi-hop questions, with agents retrieving external evidence to support their answers. For SearchQA, we follow SKILLRL [Xia et al., 2026] for the training and evaluation splits, evaluation metrics, retriever, and retrieval corpus.

H Extended Case Study Details

All cases are drawn from ALFWorld validation rollouts.

Case 1: Propose Skill. The agent was tasked with cooling a tomato and placing it on a countertop. It spent over twenty steps exhaustively sweeping cabinets and drawers, which are low-probability zones for food targets, before finally checking the fridge and countertops where the tomato was most likely to be found. The episode failed. During the subsequent skill-review phase, the agent examined the retrieved skills (Systematic Exploration, Immediate Acquisition, Destination First Policy) and recognized a gap: none of them addressed the allocation of search effort across food versus non-food zones. It called propose_skill to add Do Not Search Invalid Zones, which instructs future agents to prioritize high-probability food locations over indiscriminate sweeping.

Case 2: Update Skill. The agent was tasked with heating an apple and placing it in a sink basin. The episode succeeded, but the trajectory revealed an operational imprecision in the existing skill Open Then Heat, which instructed the agent to open the microwave, place the apple inside, and then heat it. In practice, heating failed every time the apple was placed inside first, and only succeeded when the agent held the apple in hand during the heat action. The agent called update_skill to revise this skill to Heat While Holding Target, correcting the operational guidance to keep holding the target object while executing the heat action.

Case 3: Skill Utility. The probe task belongs to the same task family as the episode from Case 2 and was selected as one of the probes during the utility reward evaluation triggered by that update. This probe task requires the agent to heat an apple and place it in a fridge. Before the revision, the skill Open Then Heat caused an 8-step confusion at the microwave: open the door, place the apple inside, attempt to heat with no effect, close and reopen the door repeatedly, take the apple back into hand, and finally heat successfully. After the revision to Heat While Holding Target, the same probe task completed the heating phase in just 2 steps: open the microwave and heat while holding the apple. Six unnecessary steps were eliminated purely by correcting the operational guidance.

I Extended Related Work

I.1 Skill Management in LLM Agents

Early work on LLM agent memory stores raw interaction histories and retrieves relevant experiences at inference time. Generative Agents [Park et al., 2023] maintain memory streams to support long-term behavior, while Reflexion [Shinn et al., 2023] uses verbal self-reflection to improve subsequent attempts. More recent memory-based agents such as Mem0 [Chhikara et al., 2025], MemRL [Zhang et al., 2026d], MemP [Fang et al., 2025], EvolveR [Wu et al., 2025b], and SimpleMem [Liu et al., 2026] further explore episodic, procedural, or lifelong memory mechanisms. These methods demonstrate the value of reusing past experience, but raw or loosely organized memories can become redundant, noisy, and difficult to maintain as experience accumulates.

A more compact alternative is to abstract trajectories into reusable *skills*. Voyager [Wang et al., 2023] maintains a skill library in Minecraft by accumulating reusable action programs, enabling open-ended exploration and reuse of past solutions. SkillRL [Xia et al., 2026] introduces a hierarchical SkillBank distilled by a teacher LLM and uses retrieved skills to guide RL training. Related work on temporal abstraction and options in reinforcement learning also motivates the use of reusable procedural units for long-horizon decision making [Sutton et al., 1999]. However, in these approaches, skill management is typically handled outside the agent's own learned policy: a teacher model, fixed rule, or predefined procedure decides what to store and when to update.

Recent concurrent work has begun to make skill management more adaptive. ARISE [Li et al., 2026d] adopts a hierarchical Manager-Worker architecture with a shared policy, where the Manager maintains a tiered skill library and selects relevant skills, while the Worker generates task solutions conditioned on those skills. MemSkill [Zhang et al., 2026a] introduces a Controller-Designer framework, where the Controller selects relevant memory skills and the Designer revises the skill set based on hard cases. COS-PLAY [Wu et al., 2025a] co-evolves a decision policy with a skill bank through an agent-managed pipeline that extracts and refines skills from unlabeled rollouts. Wang et al. [Wang et al., 2025] use RL to transform interaction experiences into reusable action-sequence skills, but rely on fixed extraction procedures rather than learned skill-management decisions. CoEvoSkills [Zhang et al., 2026b] targets autonomous construction of multi-file agent skill packages through a Skill Generator and a co-evolving Surrogate Verifier. These methods move toward adaptive skill evolution, but they still realize skill management through dedicated managerial roles, controllers, verifiers, or skill-evolution pipelines, rather than exposing skill editing as explicit self-directed decisions of the acting policy.

SKILLMASTER differs from these approaches in two ways. First, it does not delegate skill evolution to a dedicated managerial role, controller, or external skill-evolution pipeline. Instead, it applies tool-integrated reasoning to skill management: the same policy that acts in the environment also decides, through structured tool calls, whether to keep, propose, or update skills. This turns skill management into explicit self-editing behavior within the acting policy itself. Second, the value of a skill edit is not judged by surface form, heuristic triggers, or task outcome rewards alone. Instead, each candidate edit is evaluated by its counterfactual downstream utility on related probe tasks, providing an explicit skill-quality signal for RL optimization.

I.2 RL and Tool-Integrated Reasoning

RL has become a standard post-training paradigm for LLMs and LLM agents. PPO [Schulman et al., 2017] is widely used for policy optimization, and recent variants such as RLOO [Ahmadian et al., 2024], GRPO [Shao et al., 2024], DAPO [Yu et al., 2025b], and GiGPO [Feng et al., 2025] further improve stability, scalability, or group-based credit assignment. These methods have been applied to interactive settings such as embodied environments, web interaction, software engineering, and search-based reasoning [Shridhar et al., 2021, Yao et al., 2022, Yang et al., 2024, Jin et al., 2025]. Our work builds on this RL post-training paradigm but introduces a heterogeneous trajectory structure, where task-execution turns and skill-management turns carry different reward semantics.

Tool-use methods train or prompt LLMs to invoke external functions, APIs, or tools during reasoning. Toolformer [Schick et al., 2023] shows that language models can learn to use tools through self-supervision, and Gorilla [Patil et al., 2024] connects LLMs with large-scale API usage. Recent tool-integrated reasoning methods further combine tool calls with RL, including AutoTIR [Wei et al., 2025], AdaTIR [Fang and Sun, 2026], AutoTool [Zeng et al., 2026], and ASTER [Zhang et al., 2026c]. These works typically treat tools as external services that return information, perform computation, or execute actions. In contrast, the tool in SKILLMASTER edits the agent's own skill memory. This creates a distinct credit-assignment problem: the value of a write action can only be assessed by its downstream effect on future behavior. DualAdv-GRPO addresses this setting by decoupling advantage normalization for action and skill-management turns while still optimizing a single policy.

J Fair Comparison Protocol

All baseline results in Tables 1 and 2 are taken from Xia et al. [2026]. The results of SKILLMASTER in these tables are obtained from our own experiments.

K Computational Overhead

Table 3 reports the computational overhead introduced by utility-based skill management on ALFWorld. Without the utility reward, training takes on average 12.13 minutes per step, while enabling the utility reward increases the average step time to about 16.00 minutes. This corresponds to an additional 3.87 minutes per step, or roughly a 31.9% increase in wall-clock time.

Table 3: Computational overhead of utility-based skill management on ALFWorld.

Metric	Without	With
Avg. time / step	12.13 min	16.00 min
Avg. GPU memory	33.51 GiB	34.57 GiB
Peak GPU memory	61.29 GiB	70.49 GiB

We also compare GPU memory usage over the first four training steps. With the utility reward enabled, the average GPU memory allocation increases from 33.51 GiB to 34.57 GiB, while the peak allocation rises from 61.29 GiB to 70.49 GiB. These results suggest that the main computational cost of SKILLMASTER comes from the additional probe-based evaluation required by the utility reward, which increases runtime noticeably while introducing a more moderate increase in average memory usage. Overall, the utility reward introduces a noticeable runtime overhead but a relatively modest increase in average GPU memory allocation, while providing a direct learning signal for evaluating whether a candidate skill edit improves downstream performance.

L Analysis of Skill-Management Necessity

Table 4 examines two alternative explanations for the gains of SKILLMASTER: additional post-episode review and cold-start SFT. We compare GRPO with a review-only variant to assess the effect of appending a review turn, and introduce an SFT+GRPO control to assess the additional benefit of the full skill-management framework beyond a shared SFT initialization and access to the initial skill bank. Adding a post-episode review turn without enabling skill management improves success by only 0.8 percentage points on each benchmark, indicating limited gains from review alone. Full SKILLMASTER remains substantially stronger than both GRPO and its review-only variant. These results suggest that the performance gains cannot be explained merely by appending a review turn, but reflect the benefit of learning effective skill-management decisions under utility-based feedback. To further examine whether the gains can be attributed to cold-start SFT, we introduce an SFT+GRPO baseline that shares the same SFT checkpoint as SKILLMASTER. This baseline uses a static skill bank and omits post-episode skill management, counterfactual probes, and the skill-utility objective. Compared with SFT+GRPO, SKILLMASTER improves ALFWorld and WebShop success by 6.5 and 6.2 percentage points, respectively, supporting the additional contribution of the full skill-management framework beyond the shared SFT initialization.

Table 4: Analysis of skill-management necessity. ALFWorld reports overall success rate (ALL, %), and WebShop reports success rate (SUCC., %).

Method	ALFWorld	WebShop
GRPO	77.6	66.1
GRPO + Review-Only	78.4	66.9
SFT+GRPO	92.2	75.8
SKILLMASTER	98.7	82.0

M Robustness to Weaker Initial Skill Banks

Table 5 evaluates whether the gains of SKILLMASTER depend primarily on inheriting a strong initial skill bank from SKILLRL. We vary the initial skill coverage by retaining only a subset of the original skills, or removing the initial bank entirely, while keeping the same initialization for both methods. SKILLMASTER consistently outperforms SKILLRL across all settings, and the performance gap remains substantial even when the initial skill bank is heavily weakened or completely removed.

Table 5: Robustness to weaker initial skill banks. Results report ALFWorld overall success rate (ALL, %).

Initial bank	SKILLRL	SKILLMASTER
Full (100%)	89.9	98.7
Reduced (50%)	84.6	94.1
Sparse (25%)	78.8	92.5
None (0%)	74.8	89.4

Performance degrades gracefully as the initial skill bank becomes weaker, but SKILLMASTER remains clearly stronger than the teacher-driven baseline under the same initialization, including the 0% setting. These results suggest that the gains of SKILLMASTER cannot be explained solely by inheriting a strong initial skill bank. Instead, the method continues to benefit from its learned skill-management and policy adaptation even when the initial skill bank is substantially weakened or removed.

N Test-Time Skill Management

To further evaluate the test-time skill-management capability of SKILLMASTER, we test whether the trained policy can manage a fresh initial skill bank and reconstruct useful skills from an empty bank. While Section M varies the initial skill bank during SFT and RL training, the experiments here keep the trained policy parameters frozen and vary the skill bank provided at the start of evaluation and whether test-time skill management is enabled.

N.1 Skill Management from a Fresh Initial Bank

Table 6 examines whether SKILLMASTER retains its performance after removing the skill-bank changes accumulated during training. For each trained checkpoint, we compare evaluation with the training-evolved bank against evaluation with the original initial bank restored. Policy parameters remain frozen, and test-time skill management is enabled in both settings, allowing valid proposals and updates to persist across episodes. No counterfactual probes or utility feedback are used. We also include SFT+GRPO with a static initial bank as a reference. Restoring the original initial bank reduces ALFWorld success by only 0.3 percentage points. On WebShop, the score decreases by 1.9 points, while success increases by 2.7 percentage points. Compared with SFT+GRPO, the fresh-bank setting improves ALFWorld success by 6.2 percentage points, WebShop score by 5.4 points, and WebShop success by 8.9 percentage points. These results show that SKILLMASTER maintains strong performance even after the skill-bank changes accumulated during training are discarded.

Table 6: Performance with different skill banks at the start of evaluation. Results are averaged over three independent runs.

Setting	ALFWorld All	WebShop Score	WebShop Succ.
SFT+GRPO Static initial bank	92.2	87.7	75.8
SKILLMASTER Training-evolved bank	98.7	95.0	82.0
SKILLMASTER Fresh initial bank	98.4	93.1	84.7

N.2 Skill-Bank Dependence and Reconstruction

On WebShop, we further examine whether SKILLMASTER can construct useful skills from an empty bank and how its performance depends on access to the original skills.

Table 7 compares three evaluation conditions. In the first, the bank starts empty and both retrieval and skill management are disabled, so the policy acts using only the task instruction, environment observations, and current episode history. In the second, the bank starts empty but retrieval and skill management are enabled, allowing newly generated skills to be stored and reused in subsequent episodes. In the third, the policy starts with the complete original initial bank, with retrieval and skill management enabled. Enabling reconstruction from an empty bank improves the WebShop score by 6.10 points and success by 6.25 percentage points.

Table 7: Skill-bank dependence and reconstruction on WebShop using the same frozen checkpoint. Updates indicate whether test-time skill management is enabled.

Initial bank	Updates	Score	Succ.
Empty	Disabled	72.35	65.63
Empty	Enabled	78.45	71.88
Full original	Enabled	93.55	85.16

Since both conditions start without external skills, this improvement indicates that the frozen policy can construct useful skill memory from evaluation trajectories and reuse it across episodes. However, the reconstruction setting remains 15.10 score points and 13.28 success-rate points below the full-original-bank setting, where test-time skill management is also enabled. These results show that

Table 8: Performance of SKILLMASTER across three runs, reported as mean ± standard deviation.

ALFWorld							
Metric	**Pick**	**Look**	**Clean**	**Heat**	**Cool**	**Pick2**	**All**
Mean ± Std.	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	97.1 ± 0.3	95.7 ± 1.5	100.0 ± 0.0	98.7 ± 0.4

WebShop		
Metric	**Score**	**Succ.**
Mean ± Std.	95.0 ± 1.0	82.0 ± 0.8

SearchQA								
Metric	**NQ**	**TriviaQA**	**PopQA**	**HotpotQA**	**2Wiki**	**MuSiQue**	**Bamboogle**	**Overall**
Mean ± Std.	49.1 ± 0.3	62.9 ± 0.1	47.0 ± 0.1	45.6 ± 0.1	41.6 ± 0.7	20.5 ± 0.2	70.7 ± 0.2	48.0 ± 0.1

SKILLMASTER can build and use a skill bank without access to the original skills, but reconstruction only partially compensates for their removal. Explicit skill memory therefore remains important for task performance.

O Results Across Repeated Runs

Table 8 reports the mean and standard deviation of SKILLMASTER's performance over three runs on ALFWorld, WebShop, and SearchQA. We include both aggregate results and individual task-family or dataset results. On ALFWorld, SKILLMASTER achieves an overall success rate of $98.7 \pm 0.4\%$, with perfect success across all three runs on four task families. On WebShop, the average score and success rate are 95.0 ± 1.0 and $82.0 \pm 0.8\%$, respectively. On SearchQA, the overall result is $48.0 \pm 0.1\%$.

ALFWorld Skill-Review Prompt Template

```
You are reviewing a completed ALFWorld episode.
Decide whether the skill bank should be updated based on reusable evidence
from this episode.

Rules:
- Call at most one skill-management tool.
- Choose the tool that best reflects whether the episode reveals a reusable
  lesson that is missing or incorrect in the current bank.
- Use propose_skill only for a genuinely new reusable lesson.
- Use update_skill only when an existing skill should be revised.
- Use keep_skill only when the current retrieved skills already cover the
  observed strategy or failure pattern well enough.
- Base your decision on the task, episode evidence, and the current retrieved
  skills.
- Compare the observed pattern against the retrieved skills explicitly; do not
  choose keep_skill just because the broad task category already has some skills.
- Success alone is not a reason to keep_skill: if a successful episode
  demonstrates a concise reusable strategy that is not already covered,
  propose_skill or update_skill.
- Failure alone is not a reason to change the bank: if the failure does not
  reveal a concrete reusable lesson beyond the current retrieved skills, use
  keep_skill.
- For failed episodes, treat repeated invalid loops, repeated ineffective
  actions after "Nothing happens.", missed visible targets, incorrect subgoal
  switching, and losing track of required object counts as strong evidence for
  propose_skill or update_skill unless an existing retrieved skill already
  states that rule explicitly.
- For successful episodes, prefer propose_skill or update_skill when the
  success depends on a reusable tactic, ordering rule, or search heuristic that
  is not already stated explicitly in the retrieved skills.
- Add or revise a skill only if it is generic, concise, and useful for future
  ALFWorld episodes.
- Do not propose or update a skill that merely restates an already retrieved
  skill with minor wording changes.
- Only write skills grounded in the current ALFWorld household environment,
  task, objects, and receptacles.
- Do not store task-instance details such as specific object instances, room
  names, receptacle IDs, or one-off episode narration unless the lesson is
  clearly reusable.
- Do not include meta-commentary about skill-bank decisions, prompt quality,
  guidance quality, success/failure labels, or whether the bank should change.
- Do not output placeholders, ellipses, half-finished text, or copied
  trajectory fragments.
- Do not output an ALFWorld <action>; the episode is already over.
- Do not choose navigation or environment actions such as look, go to, take,
  move, clean, heat, cool, or done.
- First reason inside <think> </think>, then output exactly one skill-management
  tool call in JSON.
- Keep <think> extremely short: 1-3 sentences, no bullet points, no long
  episode recap, and no copied trajectory details.
- The JSON must be enclosed in <tool_call> </tool_call> tags.
- For task-specific skills, set category to the Skill Category shown below.
- Use category="general" only if clearly reusable across multiple task types.

Output requirements:
- Reason inside <think>...</think>, but keep it brief and decision-focused.
- Final content must be exactly one skill-management tool call wrapped in
  <tool_call>...</tool_call>.
- Inside <tool_call>...</tool_call>, output only valid JSON.
- Do not use markdown code fences.

Task: {task}
Skill Category: {category}
Episode Outcome: {outcome}
Episode Reward: {episode_reward}

Retrieved Skills:
{retrieved_skills_text}

Episode Trajectory:
{episode_trace}
END EPISODE TRAJECTORY.

Now output only the final skill-management decision using the required
<tool_call> JSON format.
```

Figure 5: ALFWorld skill-review prompt template.

WebShop Skill-Review Prompt Template

```
You are reviewing a completed WebShop episode.
Decide whether the skill bank should be updated based on reusable evidence
from this shopping episode.

Rules:
- Call at most one skill-management tool.
- Use propose_skill only for a genuinely new reusable lesson.
- Use update_skill only when an existing skill should be revised.
- Use keep_skill only when the current retrieved skills already cover the
  observed strategy or failure pattern well enough.
- Every decision must be justified by concrete evidence from THIS EPISODE, not
  by generic real-world shopping advice.
- Compare the observed pattern against the retrieved skills explicitly.
- Success alone is not a reason to keep_skill.
- Failure alone is not a reason to change the bank.
- Favor lessons about search query formulation, attribute filtering, option
  selection, product comparison, and the search-click-option-buy workflow.
- Treat missed required attributes, wrong option selection, premature buying,
  ineffective repeated searches, and losing track of constraints as strong
  evidence for propose_skill or update_skill.
- For keep_skill, the reason must tie to the actual trajectory.
- Do not default to "verify more attributes" unless trajectory evidence
  specifically supports it over a search or result-screening lesson.
- Add or revise a skill only if it is generic, concise, and useful for future
  WebShop tasks.
- Do not store one-off product titles, product IDs, page indices, exact prices,
  or episode narration unless the lesson is clearly reusable.
- Do not output a WebShop <action>; the episode is already over.
- Do not choose environment actions such as search[...], click[...], buy, etc.
- First reason inside <think> </think>, then output exactly one skill-management
  tool call in JSON.
- Keep <think> extremely short: 1-3 sentences.
- The JSON must be enclosed in <tool_call> </tool_call> tags.
- For task-specific skills, set category to the WebShop Category shown below.
- Use category="general" only if clearly reusable across multiple categories.

Shopping Task: {task}
WebShop Category: {category}
Episode Outcome: {outcome}
Episode Reward: {episode_reward}
Goal Index: {goal_idx}
Goal Text: {goal}

Retrieved Skills:
{retrieved_skills_text}

Episode Trajectory:
{episode_trace}
END EPISODE TRAJECTORY.

Now output only the final skill-management decision using the required
<tool_call> JSON format.
```

Figure 6: WebShop skill-review prompt template.

propose_skill

```
{
  "type": "function",
  "function": {
    "name": "propose_skill",
    "description": "Propose and add a new reusable skill to the skill bank
                    after a completed episode.",
    "parameters": {
      "type": "object",
      "properties": {
        "category":      {"type": "string",
                          "description": "Skill category."},
        "title":         {"type": "string",
                          "description": "Short skill title."},
        "principle":     {"type": "string",
                          "description": "Reusable principle."},
        "when_to_apply": {"type": "string",
                          "description": "Trigger condition."},
        "evidence":      {"type": "string",
                          "description": "Episode evidence."}
      },
      "required": ["category", "title", "principle",
                   "when_to_apply", "evidence"]
    }
  }
}
```

Figure 7: `propose_skill` tool schema.

update_skill

```
{
  "type": "function",
  "function": {
    "name": "update_skill",
    "description": "Update an existing skill in the skill bank after a
                    completed episode.",
    "parameters": {
      "type": "object",
      "properties": {
        "skill_id":      {"type": "string",
                          "description": "Existing skill_id or exact
                                          retrieved skill title."},
        "title":         {"type": "string",
                          "description": "Updated title."},
        "principle":     {"type": "string",
                          "description": "Updated principle."},
        "when_to_apply": {"type": "string",
                          "description": "Updated trigger condition."},
        "reason":        {"type": "string",
                          "description": "Why the skill should be revised."}
      },
      "required": ["skill_id", "title", "principle",
                   "when_to_apply", "reason"]
    }
  }
}
```

Figure 8: `update_skill` tool schema.

keep_skill

```
{
  "type": "function",
  "function": {
    "name": "keep_skill",
    "description": "Keep the skill bank unchanged after a completed episode.",
    "parameters": {
      "type": "object",
      "properties": {
        "reason": {"type": "string",
                   "description": "Why no new skill or update is needed."}
      },
      "required": ["reason"]
    }
  }
}
```

Figure 9: `keep_skill` tool schema.

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 state the paper's contributions and empirical scope: a learned skill-management policy, a counterfactual utility reward, DualAdv-GRPO, and evaluations on ALFWorld, WebShop, and SearchQA.

2. **Limitations**

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

   Answer: [Yes]

   Justification:The Conclusion and Discussion section and Appendix F discuss limitations including probe evaluation cost, probe selection heuristics, and the scope of the evaluated environments.

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 theoretical results or formal proofs.

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: The paper describes the environments, backbone model, SFT setup, optimization hyperparameters, probe selection, tool schemas, and training protocol in the experimental setup and implementation details.

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: The implementation and accompanying documentation are publicly available at `https://github.com/sduyangmin/Skill-Master`.

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: The paper specifies the training and evaluation setup, optimizer, group size, KL coefficient, learning rate, probe configuration, skill-bank format, and test-time protocol in the main setup description and implementation details.

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: Appendix O reports the mean and standard deviation of SkillMaster's performance over three runs on ALFWorld, WebShop, and SearchQA. These statistics describe run-to-run variability.

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: Hyperparameters, model configuration, SFT procedure, and training infrastructure are detailed in 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: This work evaluates LLM agents on existing benchmark environments and datasets and does not recruit human participants or conduct deceptive data collection. External assets are credited in the paper.

10. **Broader impacts**

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

    Answer: [No]

    Justification: The paper focuses on methodological development and benchmark evaluation and does not explicitly discuss potential positive and negative societal impacts.

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 submission does not release a new foundation model, scraped dataset, or other asset that poses an obvious high-risk misuse concern.

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: ALFWorld and WebShop are publicly available research benchmarks. The SFT dataset is constructed using a publicly available LLM.

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 asset is the SKILLMASTER code repository. Its README provides installation instructions, software dependencies, and example training commands.

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 studies LLM agents in simulated household and shopping environments and does not involve human subjects or personally identifiable data.

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: The work evaluates agents in controlled benchmark environments and does not involve real-world deployment or safety-critical applications.

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: LLMs are a core component of the method and training pipeline. The paper describes the backbone model, the teacher model used for cold-start SFT data generation, and the role of tool-calling in the skill-management phase.