ASK2SKILL: LEARNING TO SEEK EXPERT HELP FOR ACCUMULATING REUSABLE AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning reusable skills from interactive experience lets large language model (LLM) agents improve continually, but learning from autonomous experience is bounded by the agent's current capability, since on tasks beyond it the agent fails and yields no successful trajectory from which new skills can be distilled. We propose **Ask2Skill**, an expert-assisted skill learning framework in which the agent consults an expert at capability gaps and distills reusable skills from its own guided execution. Ask2Skill addresses how the agent should seek help and how the help it receives becomes reusable skills. First, it treats help-seeking as a learnable meta-skill informed by a textual record of the agent's capability boundary. Whether a request is necessary depends on how likely the agent is to succeed on its own, so we introduce consultation-local credit assignment, which estimates the necessity of each request from the agent's autonomous success rate at the same state and uses this estimate to shape the training signal for the meta-skill. Second, it writes successful assisted trajectories into the skill repository and, when an autonomous branch continued from the state before the request fails, contrasts the two executions to extract what turned failure into success. On ALFWorld and WebShop, Ask2Skill improves success rate over the strongest baseline by 3.8 and 4.2 points, respectively. Further analyses show that reliance on the expert declines as skills accumulate and that the skills transfer to other backbones.

1 INTRODUCTION

Large language model (LLM) agents can improve continually by learning reusable skills from interactive experience. Current research predominantly follows an experience-driven approach, in which agents accumulate execution trajectories through autonomous exploration, reflect on or summarize them (Shinn et al., 2023; Ouyang et al., 2026), and convert the results into reusable experience that guides subsequent decisions (Zhao et al., 2024; Wang et al., 2025b). However, such learning is directly constrained by the agent's current capability boundary. When a task demands knowledge, planning, or actions beyond the coverage of the existing model and skill repository, the agent not only tends to fail but also rarely obtains interaction experience that contains an effective solution process. Without such experience, the skill repository cannot cover these tasks, and the capability boundary is hard to push outward.

Reflecting on failed trajectories alone is generally insufficient to break through this bottleneck: self-reflection helps little on exploration-heavy tasks (Shinn et al., 2023), and self-correction without external feedback rarely fixes the underlying errors (Huang et al., 2024; Valmeekam et al., 2023; Kamoi et al., 2024). Failure experience can signal mistakes to avoid, but it does not necessarily supply the missing knowledge needed to make progress on the task. When humans cannot solve a problem independently, they ask a more experienced expert for hints and master new solutions by completing the task themselves. Inspired by this, we propose **Ask2Skill**, an expert-assisted skill learning framework. When autonomous capability is insufficient for the current execution, the agent proactively seeks external guidance, uses the expert's advice to continue the task, and accumulates reusable experience from its own guided execution. Ask2Skill addresses two interrelated questions. How can an agent learn to seek help from an expert at the right time and in an effective manner, and how can the incremental information obtained through help-seeking be converted into reusable skills?

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Answering the first question requires the agent to recognize when autonomous execution is blocked and accordingly decide *when to ask* and *what to ask for*. Ask2Skill therefore models this ability as a learnable *help-seeking meta-skill*, which concerns the agent's own capability rather than a particular task. Besides a fixed consultation interface, it consists of textual capability-boundary records revised with training evidence and the help-seeking behavior of a single policy that selects both environment actions and `ask_expert` requests. Training first initializes the policy on environment-verified demonstrations and then optimizes help-seeking decisions with reinforcement learning (RL). Whether a request is necessary depends on whether the agent could succeed on its own from the current state. We therefore propose *consultation-local credit assignment*, which measures the necessity of each request by the autonomous success rate from the same state. When the assisted execution succeeds, the request earns more credit the lower that rate is, and an assisted success counts slightly less than an autonomous one, so the agent prefers to act on its own when it can.

The second question requires identifying which experience in expert-assisted trajectories the agent cannot obtain through autonomous exploration. A successful assisted trajectory is still executed by the agent itself and can be written as success experience, but terminal success alone cannot tell whether the expert's information bridged a capability gap, since such a success may also occur on a task the agent could have completed on its own. For each task where help was sought, Ask2Skill therefore continues a *matched autonomous branch* from the state before the request with the consultation interface disabled. When this branch fails while the assisted trajectory succeeds, Ask2Skill further constructs contrastive experience in addition to the success experience. Because the two executions share the same prefix, this outcome difference is evidence that the expert supplied a capability the agent lacked, and the skill writer contrasts the two executions to distill the decisions that separate success from failure. The expert thus does not supply skills directly, but helps the agent overcome the bottleneck in acquiring autonomous experience.

We evaluate Ask2Skill on ALFWorld and WebShop (Shridhar et al., 2021; Yao et al., 2022). Baselines include experience and skill methods that keep the policy fixed, such as ReasoningBank and SkillOpt (Yang et al., 2026c), and RL methods with and without skill augmentation, such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and SkillRL (Xia et al., 2026). Ask2Skill outperforms both groups, improving success rate over the strongest baseline, SkillRL, by 3.8 and 4.2 points on the ALFWorld out-of-distribution split and WebShop. As skills accumulate, the agent requests help less often during construction. The repository also transfers directly to other backbones, where it remains better than those built by other methods.

Our main contributions are as follows:

- We propose Ask2Skill, which models help-seeking as a learnable meta-skill and optimizes help-seeking timing and content with textual capability boundaries and consultation-local credit based on a same-state counterfactual reference.
- We propose constructing expert-assisted experience from counterfactual trajectory pairs. Successful assisted trajectories are written into the skill repository, and when the matched autonomous branch fails, the pair is additionally written as contrastive experience, so that the repository records the capability the expert supplied.
- On ALFWorld and WebShop, Ask2Skill outperforms experience and skill methods as well as RL baselines. Ablations and analyses verify the role of each component and examine how skills accumulate during construction and transfer across backbones.

2 Related Work

Experience-driven skill learning. Many language agents learn continually by distilling reusable experience from interaction trajectories. ExpeL distills cross-task insights from successes and failures (Zhao et al., 2024). Targeting procedural knowledge, Voyager and ASI maintain retrievable libraries of programmatic skills (Wang et al., 2024; 2025a), Agent Workflow Memory and AutoManual induce reusable workflows and manuals from trajectories (Wang et al., 2025b; Chen et al., 2024), Memp consolidates experience into continually revised procedural memory (Fang et al., 2026), and ReasoningBank distills generalizable reasoning strategies from self-judged successes and failures (Ouyang et al., 2026). More recent work iteratively revises skill documents or contexts as optimizable textual states (Yang et al., 2026c; Zhang et al., 2026), or couples skill libraries

with reinforcement-learned policies (Xia et al., 2026; Shi et al., 2026; Tu et al., 2026; Yang et al., 2026a;b). In all of these methods, experience comes from interactions the agent itself has undergone. When a task exceeds the agent's current capability, the agent rarely produces successful trajectories, and failure lessons cannot supply the missing procedural knowledge. Ask2Skill uses expert guidance to extend the experience available for skill distillation beyond this boundary, while still requiring skills to come from the agent's own execution.

Learning to seek and leverage external assistance. Many methods let agents learn when to invoke resources beyond their own parameters, including inserting API calls (Schick et al., 2023), calling a code interpreter during reasoning (Feng et al., 2026), triggering retrieval on demand (Asai et al., 2024), and choosing between parametric knowledge and tools (Qian et al., 2025). When the external resource is an expert, agents can learn to request that the expert take over control (Singh et al., 2022), combine consultation with reflection and memory writing (Feng et al., 2024), or fold the consultation decision into the policy and optimize it jointly with the task objective (Song et al., 2026). Learning to defer formalizes when to hand a decision to an expert as prediction with a rejection option (Mozannar & Sontag, 2020), and recent work formulates escalation to a stronger model during generation as Bayesian optimal stopping (Shaikh, 2026). These works mainly focus on how asking helps the current task, but do not distinguish whether asking is necessary. Ask2Skill makes this distinction using same-state autonomous performance as the reference. This reference sets the credit for help-seeking in training and selects the cases for contrastive experience in skill construction, so that expert-assisted experience can help later tasks.

Metacognition and capability boundaries. Metacognition refers to monitoring and regulating one's own cognition, that is, assessing whether one can handle the current task and adjusting subsequent behavior accordingly (Flavell, 1979; Nelson & Narens, 1990). Recent work brings this framework to large language models, asking whether they can perceive and communicate their own uncertainty (Steyvers & Peters, 2026; Liu et al., 2026). In practice, metacognition in language models is commonly modeled as scalar confidence. Models predict the correctness of their own answers or are fine-tuned to refuse questions beyond their knowledge (Kadavath et al., 2022; Zhang et al., 2024). In interactive agents, such confidence also commonly gates help-seeking (Ren et al., 2023; Nguyen et al., 2019; Chi et al., 2020). Scalar confidence can decide whether to ask, but it does not indicate which capability is missing or what information to request. Another line of work explicitly annotates the scope of skills in text, reducing the risk of reusing a skill in mismatched states (Lin et al., 2026), yet these boundary records remain static. Ask2Skill uses textual records to separate situations the agent can handle autonomously from gaps that guidance can fill, and continually revises them with execution evidence gathered during training, so that the records are updated as the agent's capability changes and help-seeking is decided according to the latest capability boundary.

3 METHOD

We first formalize the agent decision process with a consultation interface (Section 3.1), then introduce the help-seeking meta-skill and capability-boundary records (Section 3.2), the training of the help-seeking policy (Section 3.3), and skill construction from counterfactual evidence (Section 3.4). Figure 1 gives an overview of the framework.

3.1 PROBLEM FORMULATION

We model the interaction between an agent and a text environment as an episodic decision process. The agent consists of a policy π_θ and an external skill repository $\mathcal{B} = \{b_i\}$, where each b_i is a textual skill distilled from prior task experience. A task specifies a goal g. At step t, let s_t denote the underlying environment state. The agent receives a textual observation o_t and the set of legal candidate actions $\mathcal{A}(s_t)$ provided by the environment. The agent's visible history h_t comprises the task goal together with all observations and actions up to the current step. Skills matching the task category, $\mathrm{Ret}_{\mathcal{B}}(g) \subseteq \mathcal{B}$, are retrieved once from the task goal at the start of the episode and injected into the prompt, and the policy generates the next action accordingly, $a_t \sim \pi_\theta(\cdot \mid h_t, \mathrm{Ret}_{\mathcal{B}}(g))$. An episode terminates when the task is completed or a fixed step limit is reached, and the task outcome is recorded as a binary success indicator $y \in \{0, 1\}$.

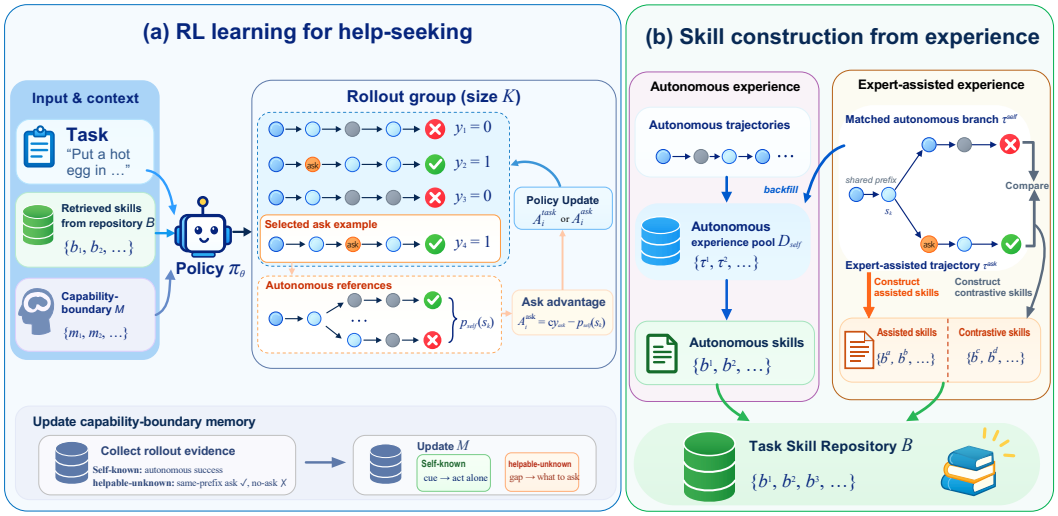


Figure 1: **Overview of Ask2Skill.** (a) During RL, the policy conditions on the task, retrieved skills, and capability-boundary memory, and chooses an environment action or a targeted expert request. Same-state autonomous continuations estimate $\hat{p}_{\text{self}}(s_k)$ for consultation-local credit, while grouped outcomes update the boundary memory with self-known and helpable-unknown records. (b) After training, matched autonomous and assisted branches share the prefix up to the request state. Autonomous trajectories and successful assisted trajectories provide ordinary experience; when an assisted success is paired with a matched autonomous failure, the writer distills the pair into contrastive experience and adds it to the skill bank.

Beyond the standard environment actions, we allow the agent to consult a frozen expert and extend the action space to $\mathcal{A}(s_t) \cup \{\texttt{ask_expert}(q)\}$, where q is a natural-language query generated by the agent. Malformed or repeated requests are handled by the environment's existing invalid-action rules. The expert replies to the query, $r_t \sim \mathcal{E}(\cdot \mid C(h_t), q_t)$, where $C(h_t)$ is the context visible to the agent. The reply r_t is appended to the context as the next observation.

3.2 Help-Seeking Meta-Skill and Capability Boundaries

Task skills in $\mathcal{B}$ describe how to solve concrete tasks, whereas deciding when to seek help and what to ask for depends on the agent's own capability. We call this ability the help-seeking *meta-skill*. Unlike a task skill, it is not a skill retrieved from $\mathcal{B}$ but has three parts. The first is a fixed interface specification, the `ask_expert` action with guidance on its use. The second is a textual memory $\mathcal{M}$ of the capability boundary, which task skills do not provide. Drawing on research on metacognition (Flavell, 1979; Nelson & Narens, 1990), its self-known records associate a class of observable situations with the autonomous policy, and its helpable-unknown records describe a capability gap, its observable trigger, and the information that should be requested. The third is the help-seeking behavior of the policy: given the interaction history, the retrieved task skills, and the boundary records $\text{Ret}_{\mathcal{M}}(g)$, it decides whether to issue a request and what targeted query to generate. Like task skills, the boundary records are retrieved once from the task goal at the start of the episode. The specification is fixed, while $\mathcal{M}$ and the policy are learned: we initialize $\mathcal{M}$ as empty and write and revise it with execution evidence gathered during training, and optimize the policy with RL (Section 3.3). Both are frozen after training, and only the task-skill repository grows during skill construction.

3.3 Training the Help-Seeking Policy

Training proceeds in two stages. Supervised fine-tuning addresses the cold-start problem (Guo et al., 2025), and RL then optimizes help-seeking decisions while revising capability boundaries.

Skill-conditioned supervised initialization. To give the agent two initial abilities, completing tasks autonomously with retrieved skills and asking the expert for help when it gets stuck, we

fine-tune on two corresponding types of execution-verified trajectories. Autonomously completed trajectories are drawn from the agent's successful samples before fine-tuning. Demonstrations of help-seeking should arise where the agent cannot succeed on its own, so expert-assisted trajectories come from tasks the agent failed autonomously. We first locate the state where the agent got stuck and generate a query and an expert reply from the prefix up to that state alone. We then replay the prefix in the environment and let the agent continue following the expert's advice, retaining only trajectories that succeed within the single-request budget. Details are in Appendix B.

Consultation-local credit assignment. The supervised stage teaches the agent to imitate the requests in the demonstrations. The RL stage must further teach it how to seek help. Standard GRPO (Shao et al., 2024), like other critic-free variants of Proximal Policy Optimization (PPO) (Schulman et al., 2017; Ahmadian et al., 2024), samples K trajectories per task and uses the group-normalized binary success reward $y_i \in \{0,1\}$ as the task advantage of all tokens, $A_i^{\text{task}} = \big(y_i - \text{mean}(\{y_j\}_{j=1}^K)\big)/\,\text{std}(\{y_j\}_{j=1}^K)$. Requests are issued mostly after a trajectory has reached a difficult state, while the group mean is set mainly by trajectories that never got stuck, so even a helpful request often ends below it. The difficulty of the state is thus counted against the request, and the policy learns to ask less. We therefore use counterfactual trajectories to measure whether a request is necessary. For each trajectory with an accepted request, we replay the execution prefix up to the pre-request state s_k, remove the consultation interface, and sample N autonomous continuations (Figure 1(a)), whose success rate serves as the *same-state counterfactual reference*, akin to Monte Carlo state-value estimates (Kazemnejad et al., 2025):

$$\hat{p}_{\text{self}}(s_k) = \frac{1}{N}\sum_{n=1}^{N} y_{\text{self}}^{(n)}, \tag{1}$$

where $y_{\text{self}}^{(n)} \in \{0,1\}$ is the task outcome of the n-th autonomous continuation.

However, starting from the same state, expert-assisted trajectories usually succeed more easily, which can encourage superfluous requests. We therefore multiply the task reward of trajectories with an accepted request by c=0.95 to express a preference for autonomous completion. On this basis, we replace A_i^{task} for the help-seeking action and its subsequent execution with

$$A_i^{\text{ask}} = c\, y_i - \hat{p}_{\text{self}}(s_{k_i}), \tag{2}$$

while tokens before the request keep A_i^{task}. When the assisted execution succeeds, this credit signal gives larger positive credit to states with lower autonomous success rates.

Evidence-driven revision of capability boundaries. Autonomous capability keeps changing during training, so $\mathcal{M}$ must be revised in step with the policy. Revision takes grouped trajectories on the same task as evidence (Figure 1(a), bottom). Tasks completed autonomously within the group provide candidate self-known records, and tasks that fail autonomously but succeed with expert assistance provide candidate helpable-unknown records. At a fixed interval, a textual updater applies add, replace, or remove operations to $\mathcal{M}$ based on these candidates, and a record is published only when at least two distinct tasks support it and no unresolved counterevidence remains. Each revision uses only evidence from the most recent interval, so stale evidence no longer takes part, and old records superseded by new evidence are updated accordingly, keeping the memory aligned with the policy's current capability. Policy updates and boundary revisions thus alternate. Evidence filtering rules and the update procedure are given in Appendix A.

3.4 Counterfactual Evidence and Skill Construction

After training is complete, the policy and capability-boundary memory are frozen, and cross-task experience accumulation is carried entirely by the skill repository (Figure 1(b)).

Autonomous experience pool. Following prior work on experience and skill learning (Zhao et al., 2024; Xia et al., 2026; Ouyang et al., 2026), we construct an autonomous experience pool $\mathcal{D}_{\text{self}}$ from the agent's autonomous trajectories, and a frozen writer $\mathcal{W}$ distills successes and failures differently. Successful trajectories are abstracted into procedural skills describing reusable procedures and their applicability conditions, and failed trajectories are summarized into preventive skills recording the

failure situation and how to avoid it. Each trajectory yields at most one candidate skill, $b = \mathcal{W}(\tau, \mathcal{B})$. Trajectories without reusable experience or duplicating existing skills are skipped (Appendix A.3). However, this pool is limited by the agent's current capability: tasks beyond it yield no successful trajectories from which the missing procedures could be distilled.

Matched autonomous branch. Expert-assisted interaction extends the source of experience beyond the agent's current capability, but once help-seeking occurs it replaces the autonomous attempt on that task, so $\mathcal{D}_{\text{self}}$ lacks autonomous experience for these tasks. For each task with an accepted request, we remove `ask_expert` from the action space under the same policy and sample one additional *matched autonomous branch*. The branch replays the assisted trajectory up to the state before the request and continues autonomously from that state. It is added to $\mathcal{D}_{\text{self}}$ as an autonomous trajectory to restore the displaced autonomous experience, which we call *matched-branch backfill*.

Expert-assisted experience construction. The writer records every successful expert-assisted trajectory τ^{ask} as success experience, $b = \mathcal{W}(\tau^{\text{ask}}, \mathcal{B})$. In addition, when the expert-assisted trajectory succeeds and its matched autonomous branch τ^{self} fails, we construct contrastive experience. Since the two executions are identical up to the request, this outcome difference arises only after it and is evidence that the expert's information supplied a capability the agent lacked at that state. The writer then receives the counterfactual pair, $b = \mathcal{W}(\tau^{\text{self}}, \tau^{\text{ask}}, \mathcal{B})$, and states the decisive difference between the two executions after the divergence state as a reusable conditional rule, takes the procedure from the assisted success, and records the failed branch's behavior as failure modes. When the matched branch also succeeds, there is no evidence that the request was needed, and only the success experience is kept. Writing rules are given in Appendix A.3.

4 Experimental Setup

Benchmarks. We evaluate on two long-horizon interactive benchmarks. ALFWorld (Shridhar et al., 2021) is a text-based household environment built on TextWorld (Côté et al., 2019) and aligned with the embodied ALFRED benchmark (Shridhar et al., 2020). Given a text goal, the agent completes one of six household task types via multi-turn interaction. Following official splits, policy training and skill construction use only the 3,553 training games, and we evaluate on the out-of-distribution UNSEEN split (134 games), reporting success rate per task type and overall. WebShop (Yao et al., 2022) is an e-commerce environment with 1.18M real products and 12k human instructions, where the agent searches, browses, and purchases a product matching the instruction. Following the official split, policy training and skill construction use only the 10,587 training instructions, and we evaluate on the 500 test instructions, reporting the average score (attribute match percentage) and success rate (percentage of fully satisfying purchases).

Baselines. Compared methods fall into two groups. Since several of them were originally evaluated on other benchmarks or with other base models, we re-implement all of them with the same base model and evaluation protocol as ours instead of reusing reported numbers. **Fixed-policy experience and skill methods** do not update policy parameters. Direct executes in a ReAct-style alternating reasoning and action manner (Yao et al., 2023) without accumulating any cross-task experience. ReasoningBank (Ouyang et al., 2026) distills retrievable reasoning memories from successful and failed trajectories. Memp (Fang et al., 2026) induces and continually revises procedural memory from experience. SkillOpt (Yang et al., 2026c) treats a skill document as an optimizable text state and iteratively edits it from scored rollouts, keeping an edit only when validation performance improves. **Reinforcement learning methods**, with or without skill augmentation, update policy parameters. GRPO (Shao et al., 2024) and Group-in-Group Policy Optimization (GiGPO) (Feng et al., 2025) adopt group-relative and group-in-group advantage estimation, respectively. EvolveR (Wu et al., 2026) alternates between distilling a principle library from its own trajectories and updating the policy with RL. SkillRL (Xia et al., 2026) co-evolves a hierarchical skill library with the policy during training.

Implementation details. The agent is Qwen3-4B (Yang et al., 2025), supervised fine-tuned and then trained with GRPO (Shao et al., 2024) (learning rate 10^{-6}, 16 tasks per batch, $K{=}8$ rollouts per task). Consultation-local credit uses $N{=}4$ same-state continuations per accepted request and an

Table 1: Main results on the ALFWorld out-of-distribution split (per-type and overall success rate, %) and WebShop (average score and success rate, %). All methods use the same base model Qwen3-4B and the same evaluation protocol. Bold indicates best in column and underline indicates second best. Complete results on the in-distribution split (SEEN) are in Appendix C.

	ALFWorld							WebShop	
Method	Pick	Clean	Heat	Cool	Look	Pick2	All	Score	Succ.
Fixed-policy experience and skill methods									
Direct (ReAct-style)	58.3	51.6	43.5	42.9	44.4	35.3	47.0	46.2	18.4
ReasoningBank	61.1	54.8	49.3	47.6	50.0	41.2	51.5	45.4	16.7
Memp	65.3	57.0	50.7	54.0	51.9	43.1	54.5	48.2	20.1
SkillOpt	69.4	62.4	55.1	50.8	57.4	49.0	58.2	51.6	23.5
Reinforcement learning methods, with or without skill augmentation									
GRPO	87.5	83.9	73.9	81.0	61.1	70.6	77.6	70.4	54.2
GiGPO	<u>91.7</u>	<u>87.1</u>	82.6	85.7	66.7	58.8	80.6	72.8	57.4
EvolveR	88.9	82.8	78.3	81.0	66.7	70.6	79.1	71.6	55.9
SkillRL	**95.8**	<u>87.1</u>	<u>87.0</u>	**93.7**	<u>72.2</u>	<u>72.5</u>	<u>85.8</u>	<u>75.2</u>	<u>61.8</u>
Ask2Skill (ours)	<u>91.7</u>	**93.5**	**91.3**	<u>90.5</u>	**85.2**	**80.4**	**89.6**	**79.2**	**66.0**

assisted-success discount c=0.95. Both the expert and the skill writer are a frozen DeepSeek-V4-Flash model (DeepSeek-AI, 2026). Repositories are built on training tasks under 3 fixed construction orders and then frozen. The consultation interface is disabled at evaluation, and we report the mean over the 3 orders. All methods decode greedily, with step limits of 50 on ALFWorld and 15 on WebShop. Prompts are given in Appendix A, and data and hyperparameters in Appendix B.

5 RESULTS

5.1 MAIN RESULTS

Table 1 reports the main results. **Ask2Skill achieves the best overall metrics on both benchmarks**, reaching 89.6% overall success on ALFWorld and 66.0% success on WebShop, 3.8 and 4.2 points above the strongest baseline, SkillRL.

Comparison with fixed-policy methods. Fixed-policy methods yield limited gains. Even the strongest of them, SkillOpt, remains far below Ask2Skill (58.2% vs. 89.6% on ALFWorld), and ReasoningBank, which distills memory from the agent's own trajectories, even falls below Direct on WebShop. These methods only modify the experience and skills in the prompt without updating policy parameters, so their gains are bounded by the execution ability of the base model.

Comparison with reinforcement learning methods. RL methods, which update policy parameters, clearly outperform fixed-policy methods. SkillRL, which adds a skill library, surpasses the strongest pure policy optimization method, GiGPO, by 5.2 points on ALFWorld, indicating that external skills and policy optimization are complementary. Since Ask2Skill also optimizes its policy with GRPO, its gain over GRPO (12.0 points on ALFWorld) can be attributed to help-seeking training and skill construction from expert-assisted experience. By task type, the gains of Ask2Skill over SkillRL concentrate on Clean, Heat, Look, and Pick2, which SkillRL has not yet solved, with the largest margins on Look and Pick2. The benefit of consultation and contrastive experience thus appears mainly on task types the agent struggles to complete on its own.

5.2 ABLATION STUDY

We conduct ablations that remove one component at a time from three groups (training, capability boundary, and skill construction) and additionally report the construction-time request rate on ALFWorld, with results in Table 2. Specifically, w/o consultation-local credit keeps the standard GRPO advantage for the request and its subsequent execution, w/o boundary memory removes the capability-boundary records, w/o matched-branch backfill uses the matched branches only for contrast, and w/o contrastive experience does not construct contrastive experience from expert-assisted trajectories and their counterfactual trajectories.

Table 3: Comparison of skill construction methods across backbones. Repositories are built on the same construction tasks and frozen. Qwen3-4B is the RL-trained Ask2Skill source policy; Qwen3-8B and Qwen3.5-27B are raw models evaluated via zero-shot skill injection (consultation interface disabled). Bold indicates best and underline indicates second best.

	Qwen3-4B (source)			**Qwen3-8B**			**Qwen3.5-27B**		
Construction method	ALFWorld	Score	Succ.	ALFWorld	Score	Succ.	ALFWorld	Score	Succ.
No skill repository	81.3	73.5	59.0	52.2	50.4	21.6	61.9	58.6	29.2
ReasoningBank writer	83.6	74.8	60.7	54.0	51.4	22.3	62.9	59.1	29.7
Self-Skill	84.3	75.6	61.8	54.5	51.9	23.1	63.4	59.9	30.5
Expert-Skill	85.1	76.0	62.1	55.2	52.5	23.9	64.2	60.3	31.1
Trace2Skill writer	<u>85.8</u>	<u>76.4</u>	<u>62.7</u>	<u>56.0</u>	<u>52.8</u>	<u>24.3</u>	<u>64.9</u>	<u>60.8</u>	<u>31.5</u>
Ask2Skill (ours)	**89.6**	**79.2**	**66.0**	**58.2**	**54.3**	**26.2**	**67.2**	**62.0**	**33.2**

Removing any component lowers success on both benchmarks. Consultation-local credit contributes the most: without it, success drops by 8.3 points on ALFWorld and 12.4 points on WebShop, and the request rate falls from 20.5% to 16.5%. We conjecture that the request rate drops because the group-relative advantage cannot tell whether a request is necessary, so even helpful requests may receive negative credit. The policy therefore asks less even in states where it needs help. Removing boundary memory lowers success by 3.0 and 1.8 points and raises the request rate to 38.2%, suggesting that boundary records mainly suppress avoidable requests. Among the skill construction components, removing contrastive experience lowers success more (3.8 and 4.6 points) than removing matched-branch backfill (2.3 and 2.6 points), while the request rate rises slightly to 24.0% and 23.8%. Without contrast, the writer cannot tell which steps depend on the expert's information; without backfill, the repository loses the agent's own experience on the tasks where help was sought. The components are thus complementary.

Table 2: Ablation results. Each row removes one component. ALF.: ALFWorld success; Web.: WebShop success; Req.: construction-time request rate on ALFWorld.

Variant	ALF.	Web.	Req. (%)
Training			
w/o consult.-local credit	81.3	53.6	16.5
Capability boundary			
w/o boundary memory	86.6	64.2	38.2
Skill construction			
w/o matched-branch backfill	87.3	63.4	23.8
w/o contrastive experience	85.8	61.4	24.0
Ask2Skill	**89.6**	**66.0**	20.5

5.3 Analysis

We examine skill gains (RQ1), cross-backbone transfer (RQ2), and construction dynamics (RQ3).

RQ1: Effectiveness of different skill construction methods. To isolate the effect of how the skill repository is constructed, we fix the trained Ask2Skill policy and vary only the construction method, with results in the Qwen3-4B (source) columns of Table 3: Self-Skill disables the consultation interface, lets the agent execute the tasks on its own, and uses our writer to construct experience from its autonomous trajectories; Expert-Skill uses the expert as the agent to execute the tasks directly and distills skills from the expert's own trajectories; in addition, the ReasoningBank and Trace2Skill (Ni et al., 2026) writers construct skills from the same trajectory set as Self-Skill. Without a repository, the frozen Ask2Skill policy is already on par with GiGPO on ALFWorld (81.3% vs. 80.6% in Table 1). On top of it, the other construction methods add only 2.3–4.5 points on ALFWorld, whereas the Ask2Skill repository adds 8.3 points; WebShop success shows the same pattern. Applied to the same autonomous trajectories, the three writers differ in gain by at most 2.2 points, so the writing method explains little of the gap. Expert-Skill, built from trajectories in which the expert completes the construction tasks itself, adds only 3.8 points, close to Self-Skill (3.0 points). Constructing experience solely from expert trajectories offers little benefit, likely because these trajectories come from a different policy and pass through states that the agent itself rarely encounters. Adding assisted successes without constructing contrastive experience (w/o contrastive experience in Table 2) also stays close to Self-Skill (85.8% vs. 84.3% on ALFWorld and 61.4% vs. 61.8% in WebShop success), whereas removing only the backfill keeps the repository above it (87.3% and 63.4%). The

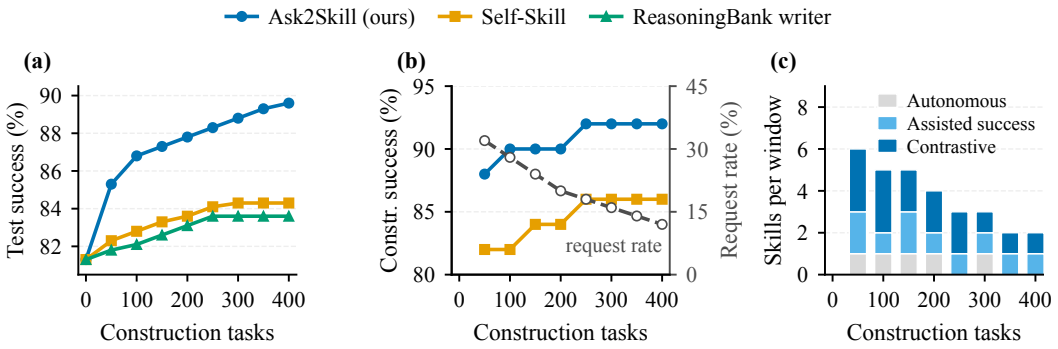


Figure 2: Skill construction dynamics on ALFWorld (mean over 3 orders). (a) Expert-free test success of repository snapshots on UNSEEN. (b) Construction-time success (left; Self-Skill without consultation) and Ask2Skill request rate (right, dashed). (c) Skills per 50-task window, by source.

gain therefore comes mainly from contrasting expert-assisted successes with the agent's own failures from the same states.

RQ2: Skill transfer across backbones. We freeze the repositories built by the Qwen3-4B policy and inject them into the prompts of Qwen3-8B (Yang et al., 2025) and Qwen3.5-27B (Qwen Team, 2026) without training, with the consultation interface disabled (Table 3). Without a repository, the two models reach 52.2% and 61.9% on ALFWorld, below the RL-trained source policy (81.3%). **Ask2Skill is best on both target backbones**: relative to no repository, it improves ALFWorld success by 6.0 and 5.3 points and WebShop success by 4.6 and 4.0 points on Qwen3-8B and Qwen3.5-27B, and outperforms Self-Skill by 3.7 and 3.8 points on ALFWorld. This shows that skills distilled from the source policy's capability gaps transfer to other models without task-specific training. The ranking of the six construction methods is also the same on all three backbones and all metrics, suggesting that the relative quality of the repositories is largely preserved across different backbones. The gains after transfer are smaller than on the source policy (8.3 points on ALFWorld) and shrink slightly as the target model grows, possibly because some capability gaps of the source policy do not exist in the target models.

RQ3: Dynamics of skill construction. Beyond completed repositories, RQ3 examines how quickly skills accumulate and how much expert help they need. We fix the Ask2Skill policy, process the 400 construction tasks in 3 orders, and evaluate repository snapshots every 50 tasks on the test set (Figure 2). First, **Ask2Skill accumulates reusable skills with high sample efficiency** (Figure 2(a)). After only 50 construction tasks, its snapshot already surpasses Self-Skill after all 400 tasks, and it keeps improving until task 350. Self-Skill plateaus after 250 tasks, consistent with autonomous experience being bounded by the agent's current capability. Second, reliance on the expert declines as skills accumulate (Figure 2(b)). The request rate falls from 32.0% in the first 50 tasks to 12.0% in the last 50 while construction-time success rises, suggesting that the repository increasingly covers states that previously required a request. Third, contrastive experience contributes the most skills, especially early on, and fewer skills are written as the repository saturates (Figure 2(c)).

6 Conclusion

We presented Ask2Skill, a framework where an LLM agent learns to consult an expert at its capability gaps and distills guided executions into reusable skills via counterfactual trajectory pairs. By treating help-seeking as a learnable meta-skill with consultation-local credit, Ask2Skill outperforms prior experience, skill, and RL baselines on ALFWorld and WebShop. With the policy fixed, repositories built from counterfactual trajectory pairs outperform those built from autonomous experience or from the expert's own trajectories. Future work includes learning from imperfect experts and multi-turn consultations.

AI USE STATEMENT

Among the tasks that require disclosure, we used generative AI tools to assist with translation and to assist in implementing the method, where AI-generated code was reviewed and tested by the authors before use. Generative models are additionally a component of the method itself. A frozen DeepSeek-V4-Flash model serves as the expert that answers help requests and as the skill writer, a language-model-based text updater revises the capability-boundary memory, and the expert replies and supplementary reasoning in the supervised corpus are likewise generated by language models. We have not used generative AI tools for developing theoretical models or conceptual frameworks, proposing or refining hypotheses, designing or giving feedback on the research methodology or the experiments, or cleaning and reformatting the dataset, and the tasks concerning mathematical claims, proofs, and qualitative or thematic analysis are not applicable to this work. Additionally, we used generative AI tools for tasks with recommended disclosure, namely searching for and summarizing related literature, and drafting and editing parts of the manuscript to improve readability. We have reviewed all AI-assisted work and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

To support reproducibility, we specify the method and the experimental protocol in full. Section 3 defines the decision process with the consultation interface, the help-seeking meta-skill and capability-boundary memory, consultation-local credit assignment, and skill construction from counterfactual trajectory pairs, and Section 4 specifies the benchmarks, the baselines, and the implementation details. Appendix A reproduces the prompts of the agent, the expert, the skill writer, and the boundary-memory updater together with the request boundary rules, and gives the complete training algorithm. Appendix B documents the construction of the supervised corpus and the supervised fine-tuning and RL configurations. Appendices C and D provide additional results and skill repository examples.

REFERENCES

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)*, pp. 12248–12267, 2024.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 589–631, 2024.

Ta-Chung Chi, Minmin Shen, Mihail Eric, Seokhwan Kim, and Dilek Hakkani-Tür. Just ask: An interactive learning framework for vision and language navigation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 2459–2466, 2020.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with IJCAI 2018*, volume 1017 of *Communications in Computer and Information Science*, pp. 41–75. Springer, 2019.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. In *International Conference on Learning Representations*, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, 2025.

Peiyuan Feng, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. AGILE: A novel reinforcement learning framework of LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

John H. Flavell. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American Psychologist*, 34(10):906–911, 1979. doi: 10.1037/0003-066X.34.10.906.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29557–29590, 2025.

Zihan Lin, Zhenyu Chen, Jiawen Wei, Xiaohan Wang, Jie Cao, Jiajun Chai, Wei Lin, Guojun Yin, and Ran He. When not to imitate: Boundary-aware skill memory for reliable tool-use LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026.

Gabrielle Kaili-May Liu, Areeb Gani, Jacqueline Lu, Jordan Thomas, Mark Steyvers, and Arman Cohan. Metacognition in LLMs: Foundations, progress, and opportunities. *arXiv preprint arXiv:2607.11881*, 2026.

Hussein Mozannar and David Sontag. Consistent estimators for learning to defer to an expert. In *International Conference on Machine Learning*, pp. 7076–7087, 2020.

Thomas O. Nelson and Louis Narens. Metamemory: A theoretical framework and new findings. In Gordon H. Bower (ed.), *The Psychology of Learning and Motivation*, volume 26, pp. 125–173. Academic Press, 1990.

Khanh Nguyen, Debadeepta Dey, Chris Brockett, and Bill Dolan. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12527–12537, 2019.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. SMART: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents. `https://qwen.ai/blog?id=qwen3.5`, February 2026.

Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 661–682, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

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

Nadeem Shaikh. Knowing when to ask for help: Bayesian self-escalation in hierarchical LLM agents. *arXiv preprint arXiv:2608.24087*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*, 2025.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10737–10746, 2020.

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

Kunal Pratap Singh, Luca Weihs, Alvaro Herrasti, Jonghyun Choi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Ask4Help: Learning to leverage an expert for embodied tasks. In *Advances in Neural Information Processing Systems*, 2022.

Siyao Song, Cong Ma, Zhihao Cheng, Shiye Lei, Minghao Li, Ying Zeng, Huaixiao Tou, and Kai Jia. EAPO: Enhancing policy optimization with on-demand expert assistance. In *International Conference on Machine Learning*, 2026.

Mark Steyvers and Megan A. K. Peters. Metacognition and uncertainty communication in humans and large language models. *Current Directions in Psychological Science*, 35(3):131–139, 2026. doi: 10.1177/09637214251391158.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, Dong Li, and Dongbin Zhao. Dynamic dual-granularity skill bank for agentic RL. *arXiv preprint arXiv:2603.28716*, 2026.

Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can large language models really improve by self-critiquing their own plans? *arXiv preprint arXiv:2310.08118*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Conference on Language Modeling*, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911, 2025b.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. From interactions to principles: Experience-driven self-distillation for evolving LLM agents. In *International Conference on Machine Learning*, 2026. Also available as arXiv:2510.16079 under the title "EvolveR: Self-Evolving LLM Agents through an Experience-Driven Lifecycle".

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Min Yang, Jinghua Piao, Xu Xia, Xiaochong Lan, Jiaju Chen, Yongshun Gong, and Yong Li. Skill-Master: Toward autonomous skill mastery in LLM agents. *arXiv preprint arXiv:2605.08693*, 2026a.

Shuo Yang, Jinyang Wu, Zhengxi Lu, Yuhao Shen, Fan Zhang, Lang Feng, Shuai Zhang, Haoran Luo, Zheng Lian, Zhengqi Wen, and Jianhua Tao. OPID: On-policy skill distillation for agentic reinforcement learning. *arXiv preprint arXiv:2606.26790*, 2026b.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026c.

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

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

Hanning Zhang, Shizhe Diao, Yong Lin, Yi R. Fung, Qing Lian, Xingyao Wang, Yangyi Chen, Heng Ji, and Tong Zhang. R-Tuning: Instructing large language models to say 'I don't know'. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 7113–7139, 2024.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Prompts and Protocol Details

This appendix gives the prompts of the model-facing components in Section 3 and the training procedure of the RL stage. Appendices A.1 and A.2 give the consultation description in the agent prompt and the expert prompt. Appendices A.3–A.5 give the skill writer, the capability-boundary memory, and the memory updater. Appendix A.6 gives the help-annotation prompts for the supervised corpus, and Appendix A.7 gives the training procedure.

A.1 Consultation Description in the Agent Prompt

In ALFWorld, the fixed interface specification of the help-seeking meta-skill (Section 3.2) is the following block.

ALFWorld Agent Prompt Fragment: Expert Consultation Meta-Skill

```
### Meta-Skill: Expert Consultation
In addition to ordinary environment actions, you may ask the external
expert for help at most once in an episode by taking an action of the
form:
<action>ask_expert: ...</action>

Use this action only when expert help is useful for completing the
task. The expert returns guidance as the next observation. After
receiving that guidance, continue with ordinary environment actions.
```

WebShop uses the following block, placed after the retrieved skills. The prompt of each step also states whether the consultation has been used.

WebShop Agent Prompt Fragment: Expert Consultation

```
## Optional Expert Consultation (one-shot)
You may use ask_expert: <xxx> at most once per task when a specific
uncertainty about the task, the current page, or a recovery strategy
cannot be resolved reliably from the visible evidence and retrieved
skills. Otherwise, continue independently with an ordinary WebShop
action. Replace <xxx> with one focused question grounded in the
visible task, page, and relevant history.

Consultation does not change the page or consume an ordinary WebShop
action. The expert provides a short hint for your next decision. It
may suggest an action, but it does not execute or force any action;
independently choose one currently legal ordinary action using the
hint and visible evidence.

After consultation, continue with ordinary actions and do not ask
again, even if the ASK template remains listed. Your question is not
repeated in subsequent action prompts.
```

A.2 Expert

The expert is a frozen DeepSeek-V4-Flash model. Once a request is accepted, the expert replies once based on the request and the context currently visible to the agent. ALFWorld uses the following system prompt.

ALFWorld Expert System Prompt

```
You are a frozen expert helping an ALFWorld agent. You receive the
student's request and only runtime-visible state: task goal, bounded
recent history, current observation, and current native admissible
actions. Ground every claim in that context. Prefer concise legal
next actions when possible. Do not invent object locations, room
structure, action syntax, hidden simulator state, or future
observations. If the visible state is insufficient, give a systematic
legal search procedure rather than guessing.
```

The policy alternates reasoning and actions in the ReAct format (Yao et al., 2023). A request is parsed from the action string. In ALFWorld, the reply to an accepted request is appended as the next observation with the prefix `Expert Response:` and is excluded from the policy loss.

WebShop uses the following system prompt. The JSON object returned by the expert is rendered as text giving the suggested action with its basis and the task requirements still to be confirmed.

WebShop Expert System Prompt

```
You are a frozen expert helping a WebShop agent. You receive only its
runtime-visible state: task goal, bounded recent history, current
observation, and currently admissible ordinary actions. Ground every
claim in that payload.

Your answer has two parts. "immediate" is the move to make right now
on the page you can see; the action must be legal right now, so a
click must exactly match a supplied click action, and a search is
allowed only when the supplied list contains a free-text search
action. "standing" is what remains true for this task no matter which
page the student reaches later, so it must contain no control names,
no product identifiers, no ordered steps, and nothing that is only
true of the current page---only the task's own outstanding
requirements and judgement rules.

Never state that an unshown size, color, material, feature, or other
requested attribute matches. Do not use hidden target metadata or
future outcomes.

Return only one JSON object:
{"immediate": {"recommended_action": "<search[...] or exact
click[...]>",
   "basis": "<which visible clue makes it the right move>"},
 "standing": {"open_constraints": ["<task requirement not yet
confirmed>"],
   "decision_rules": ["<if/then rule valid on any page>"]}}
```

A.3 Skill Writer

The skill writer builds the task-skill repository of Section 3.4. It receives one scored trajectory in compacted form, the titles of existing skills, and a source label, and constructs at most one skill. ALFWorld and WebShop use the two system prompts below.

ALFWorld Skill Writer System Prompt

```
You compile one trajectory into at most one reusable ALFWorld
procedural skill.

Use only the supplied task goal and policy-visible trajectory.
Generalize away object instance numbers, exact room locations, and
task-specific names. Never mention ASK_EXPERT, an expert, an expert
response, a hidden state, a game file, reward, or future/test tasks.
A success may yield a reusable procedure; a failure may yield a
reusable prevention or systematic-search rule. Do not restate an
```

```
existing skill.  If the trajectory contains no defensible reusable
lesson, skip.

Return exactly one JSON object and no markdown.  Either:
{"decision":"write","title":"3-8 words",
  "principle":"...","when_to_apply":"...",
  "procedure":"...","preconditions":"...",
  "failure_modes":"..."}
or:
{"decision":"skip","reason":"short source-local reason"}
```

WebShop Skill Writer System Prompt

```
You compile one trajectory into at most one reusable WebShop shopping
skill.

Use only the supplied shopping instruction and policy-visible
trajectory.  A useful skill concerns search query formulation, product
selection, option configuration, constraint verification, navigation
between result and product pages, or price handling.  Generalize away
product identifiers (ASINs), exact product titles, brand names, exact
prices, and option values that belong to one instruction; keep the
kind of requirement instead (for example, a required size, color,
material, pack count, or price ceiling).  Never mention ASK_EXPERT, an
expert, an expert response, a hint, a hidden target product, reward,
score, or future/test tasks.  Do not restate an existing skill.  A
success may yield a reusable procedure; a failure may yield a reusable
prevention or verification rule.  If the trajectory contains no
defensible reusable lesson, skip.

Return exactly one JSON object and no markdown.  Either:
{"decision":"write","title":"3-8 words",
  "principle":"...","when_to_apply":"...",
  "procedure":"...","preconditions":"...",
  "failure_modes":"..."}
or:
{"decision":"skip","reason":"short source-local reason"}
```

The constructed skill must be a procedure the agent can carry out on its own, rather than a copy of the expert's advice. Allowing the writer to skip prevents it from adding skills similar to existing ones. The repository is constructed in batches. The writer processes one batch of trajectories at a time, and each batch may add only a limited number of new skills, controlling repository size and redundancy. When the matched autonomous branch fails, the contrastive writer additionally receives the assisted success (`trajectory`) together with the failed branch (`paired_failure`). The matched branch is replayed by the same policy up to the state before the request and continues from that state with `ask_expert` removed from the action space, so the two views share their prefix up to a divergence step. The writer first identifies the decisive difference between the two views after this step and states it as a conditional rule in the principle and applicability fields. The procedure is written only from the successful trajectory, and what the failed branch did, and how it failed, goes only into the failure-modes field. The prompt of the contrastive writer is shown below.

ALFWorld Contrastive Writer System Prompt

```
You compile a paired counterfactual into at most one reusable ALFWorld
procedural skill.

The request carries two views, each labelled with its own outcome:
  • "trajectory":  the reference episode, which succeeded.
  • "paired_failure":  an episode identical to the reference up to a
shared divergence step that continues differently afterwards.  Its
result field states how it ended.

Compare the two views, and attribute every claim to the view that
supports it:
```

```
1. Starting from the divergence step, identify the decisive
difference:  the earliest decision at which the reference acted
differently from "paired_failure" and its own observations show
progress that "paired_failure" never made.
2. principle and when_to_apply state this difference as a reusable
conditional rule:  the observable situation at the divergence step,
the decision the reference made there, and the alternative that
"paired_failure" took instead.
3. procedure and preconditions describe only the REFERENCE
("trajectory") episode.  State its procedure as the actions it
actually carried out, in order.  Do not add a step the reference
episode did not perform.  In particular, do not require a precondition
the reference did not need, and do not claim a quantity of objects was
moved if the episode's own environment messages confirm a different
number.  When the reference succeeded because part of the goal was
already satisfied, say so rather than generalising it into a rule the
reference never demonstrated.
4. failure_modes describes what "paired_failure" did after the
divergence step and how it ended.  Never put "paired_failure" actions
into procedure.
5. Only claim simultaneous possession of two objects if an inventory
observation shows both at once.  Placing two objects one after another
does not show they were held together.
6. Add "evidence" citing the decision_index values you relied on,
taken from the step list of each view:  {"success_steps":[...],
"failure_steps":[...]}.  Cite only indices that appear in that view,
and include the step of each view at the decisive difference.

Generalize away object instance numbers, exact room locations, and
task-specific names.  Never mention ASK_EXPERT, an expert, an expert
response, a hidden state, a game file, reward, or future/test tasks.
Do not restate an existing skill.  If the views show no defensible
reusable difference, skip.

Return exactly one JSON object and no markdown.  Either:
{"decision":"write","title":"3-8 words",
  "principle":"...","when_to_apply":"...",
  "procedure":"...","preconditions":"...",
  "failure_modes":"...",
  "evidence":{"success_steps":[...],"failure_steps":[...]}}
or:
{"decision":"skip","reason":"short source-local reason"}
```

The contrastive writer prompt used in WebShop replaces rules 3 and 5 and the generalization requirement, as shown below.

WebShop Contrastive Writer System Prompt (Changed Parts)

```
3. procedure and preconditions describe only the REFERENCE
("trajectory") episode.  State its procedure as the actions it
actually carried out, in order:  the searches it issued, the result
and product pages it opened, the options it selected, and when it
bought.  Do not add a step the reference episode did not perform, and
do not require a check the reference did not make.
5. Only claim that a product satisfies a requirement if a page
observation in that view shows it.  A title or a results-page entry is
evidence only for the words and prices it displays, and an option
counts as selected only if the episode clicked it on the product page
before buying.
```

WebShop Contrastive Writer System Prompt (continued)

```
Generalize away product identifiers (ASINs), exact product titles,
brand names, exact prices, and option values that belong to one
instruction; keep the kind of requirement instead.  Never mention
ASK_EXPERT, an expert, an expert response, a hint, a hidden target
product, reward, score, or future/test tasks.  Do not restate an
existing skill.  If the views show no defensible reusable difference,
skip.
```

A.4 Textual Capability-Boundary Memory

The memory of Section 3.2 admits exactly two record types. In the template below, text in angle brackets describes the content of each field.

```
[Self-known]
Scope:  <task/subgoal/state pattern>
Observable cue:  <runtime-visible evidence that autonomous execution
is sufficient>
Autonomous strategy:  <brief strategy; continue without asking>

[helpable-unknown]
Capability gap:  <specific unresolved capability deficit or
subproblem>
Observable trigger:  <runtime-visible evidence of the capability gap>
Ask strategy:  <what information or procedure to request>
```

The memory starts empty and is written only by the updater (Appendix A.5). Retrieved records are injected into the policy prompt, and the retrieval count, capacity, and revision interval are given in Table 5. Every record written by the updater must be supported by rollouts. Before each revision, we select by rule, from the rollouts of the most recent I training steps, those that may serve as evidence, using the following conditions for the two record types.

- **Self-known**: the autonomous rollout on the task succeeded.
- **helpable-unknown**: the task has a verified same-prefix comparison in which the consumed expert reply led to success and the matched autonomous continuation failed.

The updater proposes edits based on these rollouts, and each proposed edit is then validated. An add or replace must cite supporting rollouts from at least two distinct tasks, all eligible for the record type. A remove must cite counterevidence. A call may make at most two edits.

A.5 Memory Updater

The updater is a frozen DeepSeek-V4-Flash model queried with temperature 0 and at most 4,096 output tokens. It receives the current memory, the supplied rollouts with an outcome index over them, and the evidence rollouts selected as in Appendix A.4. ALFWorld uses the following system prompt.

ALFWorld Memory Updater System Prompt

```
You are the Help-Seeking Meta-Skill Writer.  From the supplied student
trajectories, infer narrowly scoped capability boundaries and record
them as Self-known or helpable-unknown records.

## Record types (three fields each)
- Self-known:  Scope; Observable cue; Autonomous strategy.
- helpable-unknown:  Capability gap; Observable trigger; Ask strategy.

## Evidence rules
1.  Read the outcome index first and copy paired outcomes exactly:
help1_self1 means both branches succeeded; a missing pair is unknown,
not a failure.
```

```
2.  Infer held objects, searched locations, and task progress from
environment feedback on native actions, never from the student's
queries or the expert's advice.  "Not found in the trace" does not
mean "absent from the environment".
3.  A Self-known record needs successful autonomous trajectories on
two distinct tasks that actually show the stated cue.
4.  A helpable-unknown record needs two distinct tasks from its
eligibility list, each with a verified same-prefix comparison in which
the consumed help succeeds and the no-help branch fails, sharing the
trigger before the request.  Help success alone is insufficient.
5.  Check failures as well as successes.  If an unresolved
counterexample applies, narrow the scope or withhold the record.

## Writing rules
- Abstract each trajectory into observable subgoal transitions,
compare them across tasks, and state a narrow cue with a short
reusable conditional strategy.
- Remove object instance numbers, exact locations, and task-specific
names; do not write fixed action sequences for a single task.
- Keep each field to about one sentence and never put evidence IDs in
the fields.

## Edit rules
- Make at most two edits (add, replace, or remove); an empty list is
valid.
- Replace or remove an existing record whose condition is contradicted
by failures; a remove must cite counterevidence.

## Output format
Return only one JSON object:
{"assessment":  "supporting and opposing evidence",
  "edits":  [{"operation":  "add|replace|remove",
    "record_id":  "required for replace/remove",
    "record_type":  "Self-known|helpable-unknown",
    "fields":  {...}, "support_ids":  ["E000", "E001"],
    "counter_ids":  [], "reason":  "evidence justification"}]}
For add and replace, counter_ids must be empty.
```

The response must be a single JSON object. Edits that fail the validation of Appendix A.4 are not applied.

The updater prompt used in WebShop replaces evidence rule 2 and the writing rules, as shown below. In its trajectory view, product identifiers in the admissible-action lists are replaced by `<asin>`, free-text search is shown as `search[<query>]`, and the transition index marks search, product-page, option, and purchase steps.

WebShop Memory Updater System Prompt (Changed Parts)

```
## Evidence rule 2
Student actions, especially text inside ASK queries, may contain false
claims about which product was opened, which options were selected, or
whether a requirement is satisfied.  Infer these facts from the page
observations that follow native actions, never from the student's
self-description or the expert's advice.  A requirement that is not
shown on any page in the trace is unverified, not absent from the
product.

## Writing rules
- Abstract each trajectory into observable shopping transitions
(search issued, results inspected, product page opened, option
selected, requirement checked, purchase), compare them across tasks,
and state a narrow cue with a short reusable conditional strategy.
- Remove product identifiers, exact product titles, brand names, exact
prices, and option values that belong to one instruction; do not write
fixed action sequences for a single instruction.
```

```
- Distinguish refining a search or opening a new candidate from
repeating the same search or reopening the same page without new
observations.
- Keep each field to about one sentence and never put evidence IDs in
the fields.
```

A.6 HELP-ANNOTATION PROMPTS FOR THE SUPERVISED CORPUS

The help annotation of Appendix B.1 uses two independent calls. A request-point selector reads the complete failed trajectory and chooses where to ask, and a query writer that sees only the visible state at the chosen point writes the query.

Request-point selector. The selector used in ALFWorld returns the earliest state with a specific information or decision gap, or declines when no such state exists. The selector used in WebShop receives the task goal, every ordinary action with the admissible actions before it, and a per-step error label derived from observable feedback, and proposes up to three candidate request states; the questions it writes are discarded.

ALFWorld Request-Point Selector System Prompt

```
You annotate ONE ALFWorld failed autonomous trajectory.  Choose the
earliest runtime state where one expert consultation would plausibly
increase the chance of completing the task.  You may inspect the full
failed trajectory OFFLINE only to decide the boundary.

State j means exactly j ordinary actions have already executed.  State
0 is the initial observation.  The observation produced by action j-1
belongs to State j.  Do not ask merely because the episode eventually
fails:  choose a state with a specific information/decision gap that
an expert could address.  If no such state exists, return
needs_help=false.

Return exactly one JSON object:
{"needs_help":  true/false, "actions_executed":  0,
  "difficulty_mode":  "initial_gap" or "mid_trajectory_gap",
  "diagnosis":  "short offline-only reason"}
```

WebShop Request-Point Selector System Prompt

```
You place help interventions in a FAILED WebShop trajectory.

You receive one complete failed autonomous trajectory:  the task goal,
every ordinary action with its pre-state admissible actions, and a
per-step error label derived only from observable evidence.  You
decide WHERE a single expert consultation would plausibly have changed
the outcome, AND WHAT the student should ask there.

State k means the state after exactly k ordinary actions have been
executed.

Choose up to 3 intervention points, at most one per distinct k,
ordered from most to least likely to change the outcome.  Placements
worth considering, none mandatory:
- strictly BEFORE an action that went wrong, so the student can ask
first;
- inside a state already gone wrong, so the student can ask how to
recover;
- immediately before committing a purchase, to ask for verification;
- at a first results page or a stalled page, when the route itself is
unclear.
Prefer points where the observed error shows the student lacked
something that a short piece of advice could supply.

For each point, write the question the STUDENT would ask the expert at
State k.
```

```
Ground the question ONLY in what is visible at or before State k: the
task goal, the observation and admissible actions at State k, and the
earlier history. The trajectory continues past State k and you have
seen it; you must not use any of that. Never mention a product
detail, option value, price or product id that is not already visible
at or before State k. Do not request the hidden answer, do not ask
the expert to decide the whole task, and do not include action or
think tags.
Return only one JSON object:
{"points": [{"actions_executed": <integer k>, "question": "<one
question>",
  "decision_basis": "<one short sentence>"}]}
Return an empty list when no intervention could plausibly have helped.
```

Query writer. The query writer receives only the runtime-visible state at the selected point. In WebShop, it is called once for each candidate state.

ALFWorld Query Writer System Prompt

```
Write one concise ALFWorld expert consultation action at the supplied
runtime-visible state. The only special action is:
ask_expert: <free-form request>
Use ONLY the payload: goal, current observation, recent observed
actions and observations, and current native admissible actions. Do
not invent an object, location, inventory fact, completed action, or
hidden state. Ask for guidance that can change the next decision.
Prefer a concrete question over a generic request for help. Keep the
request short enough for a 256-token student budget.
Return exactly one JSON object:
{"ask_action":"ask_expert: ..."}
```

WebShop Query Writer System Prompt

```
Write one natural ASK_EXPERT request for a WebShop student.
You receive only the runtime-visible State k: task goal, recent
observed history/actions, current observation, and current ordinary
actions. You cannot see the failure suffix, hidden target product,
future reward, or selector diagnosis. Ask for the specific diagnosis
or strategy needed to continue. The request may ask how to search,
what evidence to inspect, how to compare candidates, how to choose
visible options, or when a purchase is justified. Do not assert
unobserved facts, answer the shopping task yourself, include
action/think tags, or request a second consultation.
Return only one JSON object: {"query": "<non-empty request>"}.
```

A.7 Training Procedure

Algorithm 1 summarizes the RL stage of Section 3.3, and Table 5 gives its configuration. It uses two subroutines. REPLAY replays the prefix of τ_i up to s_{k_i} in the environment, removes `ask_expert` from the action space, samples N autonomous continuations, and returns their task outcomes. REVISE passes the current memory, the rollouts $\mathcal{R}$ collected since the previous revision, and the evidence rollouts selected from them (Appendix A.4) to the memory updater (Appendix A.5), and applies at most two validated edits.

Algorithm 1 RL of the help-seeking policy

Require: supervised policy π_θ; frozen expert $\mathcal{E}$; frozen task-skill repository $\mathcal{B}$; training tasks $\mathcal{D}_{\text{train}}$; group size K; continuations N; assisted-success discount c; revision interval I; training steps U

Ensure: trained policy π_θ; boundary memory $\mathcal{M}$

$\mathcal{M} \leftarrow \emptyset, \mathcal{R} \leftarrow \emptyset$
for $u = 1, \ldots, U$ **do**
  ▷ *Rollout with the consultation interface*
  Sample a batch of tasks from $\mathcal{D}_{\text{train}}$
  Roll out $\{\tau_i\}_{i=1}^{K}$ per task with $a_t \sim \pi_\theta(\cdot \mid h_t, \mathrm{Ret}_{\mathcal{B}}(g), \mathrm{Ret}_{\mathcal{M}}(g))$
  Answer each accepted $\texttt{ask_expert}(q_t)$ with $r_t \sim \mathcal{E}(\cdot \mid C(h_t), q_t)$
  Compute y_i and A_i^{task}, with the rewards of assisted trajectories scaled by c
  ▷ *Consultation-local credit*
  for each τ_i with an accepted request at state s_{k_i} **do**
    $\{y_{\text{self}}^{(n)}\}_{n=1}^{N} \leftarrow \textsc{Replay}(\tau_i, s_{k_i}, N)$
    $\hat{p}_{\text{self}}(s_{k_i}) \leftarrow \frac{1}{N}\sum_{n=1}^{N} y_{\text{self}}^{(n)}$ ▷ Eq. 1
    $A_i^{\text{ask}} \leftarrow c\, y_i - \hat{p}_{\text{self}}(s_{k_i})$ ▷ Eq. 2
  end for
  ▷ *Policy update*
  Update π_θ by GRPO with A_i^{ask} from the request onward and A_i^{task} before it
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\tau_i\}$
  ▷ *Boundary revision*
  if $u \bmod I = 0$ **then**
    $\mathcal{M} \leftarrow \textsc{Revise}(\mathcal{M}, \mathcal{R}), \mathcal{R} \leftarrow \emptyset$ ▷ Appendix A.5
  end if
end for
return π_θ, $\mathcal{M}$

B Corpus Construction and Training Details

This appendix gives the supervised corpus (Appendix B.1) and the training configuration (Appendix B.2) used by the two training stages in Section 3.3.

B.1 Help-Capable Supervised Corpus

The cold-start corpora for ALFWorld and WebShop are built with the same procedure, and the prompts used in each environment are given in Appendix A.

Autonomous trajectories. We collect trajectories on training tasks, distill the successful and failed ones into a skill repository, and freeze it. We then select the collected trajectories that complete their tasks and keep their executed action sequences as autonomous trajectories. The prompts of the training examples contain skills retrieved from the frozen repository.

Expert-assisted trajectories. We construct expert-assisted trajectories from autonomous failures. A selector reads the complete failed trajectory and determines the difficult state at which to ask. The trajectory is truncated there, and we write the query from the truncated prefix, so that the query does not depend on the failure that follows. After the frozen expert replies, the prefix is replayed in the environment and the agent continues from the reply. We keep only the trajectories that finally succeed with expert assistance.

Composition. Both kinds of trajectories are flattened into step-level supervision samples, each taking the context before one decision as input and the action of that step as the target. Table 4 gives the corpus size of both environments by source and by split.

Table 4: Supervised corpus size. A step-level supervision sample corresponds to one decision of a successful trajectory. The train and validation splits are task-disjoint.

	ALFWorld			WebShop		
	#Tasks	#Traj.	#Samples	#Tasks	#Traj.	#Samples
By source						
Autonomous	346	507	6,609	317	317	1,482
Expert-assisted	165	165	2,002	319	319	2,419
Total	511	672	8,611	636	636	3,901
By split						
Train	487	642	8,178	604	604	3,704
Validation	24	30	433	32	32	197

B.2 Training Configuration

The policy is Qwen3-4B (Yang et al., 2025), trained in the verl framework (Sheng et al., 2025). Table 5 lists the configuration of both stages in both environments. The RL settings are shared by all variants, ALFWorld draws RL tasks from all 3,553 official training tasks, and WebShop draws them from the 10,587 instructions of the official training split. Supervised fine-tuning runs on four NVIDIA H100 80GB GPUs. All RL runs use random seed 42.

Table 5: Training configuration. Symbols follow Section 3.3; a hyperparameter with a single value is the same in both environments.

Hyperparameter	ALFWorld	WebShop
Supervised fine-tuning		
Learning rate	1×10^{-5}	1×10^{-4}
Global batch size	32	16
Epochs	1	3
Max sequence length	2,816	16,384
Weight decay	0.01	
Warmup ratio / schedule	0.03 / cosine	
RL (GRPO (Shao et al., 2024))		
Initialization	supervised checkpoint	
Learning rate	1×10^{-6}	
Batch size	16 tasks $\times$ K=8 rollouts	
Training steps	32	
KL coefficient	0.01	
Sampling temperature	0.7	
Max response length	512 tokens	
Task reward (success / assisted success / failure)	1, c, 0	
Invalid-action penalty coefficient	0.1	
Help-seeking and boundary memory		
Assisted-success discount c	0.95	
Same-state continuations N	4	
Memory capacity	24 records	
Memory retrieval top-k	2	
Revision interval I	4 steps	
Max edits per revision	2	

C Additional Results

Table 6 reports results on the ALFWorld in-distribution split (SEEN) under the same evaluation setting as Table 1.

Across all methods, performance on the in-distribution split (SEEN) is consistently higher than on the out-of-distribution split (UNSEEN). Consistent with Table 1, Ask2Skill achieves the highest overall success rate of 92.9% (130/140 tasks), outperforming SkillRL by 2.9 percentage points (+4 tasks). While SkillRL reaches near-saturation on simpler single-step retrieval tasks such as Pick

Table 6: Per-type and overall success rate (%) on the ALFWorld in-distribution split (SEEN, 140 tasks). The evaluation setting is the same as in Table 1. Bold indicates best in column and underline indicates second best.

Method	Pick	Clean	Heat	Cool	Look	Pick2	All
Direct (ReAct-style)	65.7	55.6	50.0	48.0	53.8	41.7	53.6
ReasoningBank	71.4	59.3	56.2	52.0	61.5	41.7	57.9
Memp	74.3	63.0	62.5	56.0	61.5	45.8	61.4
SkillOpt	77.1	66.7	68.8	60.0	69.2	50.0	65.7
GRPO	91.4	85.2	75.0	84.0	69.2	75.0	82.1
GiGPO	<u>94.3</u>	<u>88.9</u>	<u>87.5</u>	88.0	76.9	70.8	85.7
EvolveR	91.4	85.2	81.2	88.0	76.9	75.0	84.3
SkillRL	**97.1**	<u>88.9</u>	<u>87.5</u>	**96.0**	<u>84.6</u>	<u>79.2</u>	<u>90.0</u>
Ask2Skill (ours)	<u>94.3</u>	**96.3**	**93.8**	<u>92.0</u>	**92.3**	**87.5**	**92.9**

(97.1%), Ask2Skill delivers its decisive advantages on multi-step, failure-prone categories including Clean (+7.4 points), Heat (+6.3 points), Look (+7.7 points), and Pick2 (+8.3 points), verifying that targeted contrastive error distillation predominantly benefits states that agents struggle to resolve autonomously.

D SKILL REPOSITORY EXAMPLES

This appendix contrasts skills written from autonomous trajectories, the only trajectories Self-Skill writes, with skills written from trajectories that succeed with expert assistance, which Ask2Skill additionally writes. We use two task types, heating and placing two objects. All entries come from repositories built with the same policy and writer and are quoted verbatim.

Heating. Both entries come from the same construction run and were written on two tasks with the goal "heat some mug and put it in cabinet". The first comes from an autonomous success. The agent opened the microwave, four cabinets, and the fridge without finding a mug, then found one on a countertop, heated it, and put it in a cabinet. The writer therefore tells the agent to search surfaces and lists searching only closed containers as a failure mode. The second comes from an expert-assisted success. The agent checked the toaster, the stoveburners, the sink basin, and the microwave without finding a mug and then asked for help. The expert replied that mugs are commonly stored in cabinets and drawers and suggested checking them in turn, and the agent found the mug in the first cabinet, heated it, and put it in a cabinet. The resulting skill adds what the autonomous skill lacks, namely where to search when the mug is not on an open surface, and it places the heated mug in the destination cabinet required by the goal.

Entry Written from an Autonomous Trajectory (Heating)

title: Heat mug then place in cabinet

principle: For tasks requiring a heated object to be stored in a cabinet, locate the object, heat it in a microwave, then place it in an open cabinet.

when_to_apply: When the goal is to heat a mug (or similar object) and put it in a cabinet.

procedure: 1. Search surfaces and open containers to find the mug. 2. Pick up the mug. 3. Go to a microwave and open it if closed. 4. Heat the mug with the microwave. 5. Go to a cabinet and open it if closed. 6. Move the mug into the cabinet.

preconditions: The mug is reachable and not already heated. A microwave and at least one cabinet are available.

failure_modes: Attempting to place the mug in a closed cabinet; heating the mug before picking it up; searching only closed containers and missing the mug on a surface.

Entry Written from an Expert-Assisted Trajectory (Heating)

title: Search cabinets for mug before heating

principle: Mugs are commonly stored in closed cabinets and drawers; when a heat-and-place task requires a mug and it is not visible on open surfaces, search closed storage receptacles systematically before checking other appliance interiors.

when_to_apply: When the goal requires heating a mug and placing it in a cabinet, and the mug is not found on visible surfaces or in the microwave.

procedure: 1. Open and inspect the microwave to confirm whether the target mug is inside; if not, proceed. 2. Search closed cabinets and drawers one by one, opening each and checking contents. 3. When the mug is found, take it. 4. Carry the mug to the microwave and heat it. 5. Carry the heated mug to the destination cabinet and place it inside.

preconditions: A microwave is available for heating. At least one cabinet or drawer is present and closed. The mug is not already held or visible on open surfaces.

failure_modes: Searching open surfaces or appliances repeatedly instead of opening closed storage. Attempting to heat the mug before acquiring it. Placing the mug in a closed cabinet without opening it first. Leaving the microwave occupied by another object that blocks heating.

Placing two objects. Both entries come from one construction run, which uses a different task order from the heating example. The first comes from an autonomous success on "put two spraybottle in drawer". The agent found one spraybottle on the toilet and put it in the drawer, then found the second on a countertop. Its skill searches the surfaces where such objects usually lie and places each object as soon as it is found. The second comes from an expert-assisted success on "find two pencil and put them in coffeetable". The agent found the first pencil on a side table and placed it, then asked where to search for the second. The expert replied that pencils are commonly stored in drawers and on the dresser, and after checking three drawers and the garbage can, the agent found the second pencil on the dresser. The resulting skill tells the agent where to continue searching once the first instance is placed and lists giving up before checking dressers and drawers as a failure mode.

Entry Written from an Autonomous Trajectory (Two Objects)

title: Search toilet then countertops for spraybottles

principle: When a task requires collecting two instances of the same object type, search likely container surfaces one by one, taking the first instance to the destination before continuing the search for the second.

when_to_apply: Tasks requiring two identical objects to be placed in a container, especially when the first search location yields only one instance.

procedure: 1. Scan the room for likely object surfaces (toilets, countertops, sinks, tables). 2. Go to the first likely surface and take any matching object found. 3. Carry it to the target container, open the container if closed, and place the object inside. 4. Continue searching remaining likely surfaces for a second matching object. 5. Take the second object and place it in the same target container.

preconditions: The target container is known and reachable; at least one matching object is visible on a surface.

failure_modes: Assuming all matching objects are in one location; forgetting to open a closed target container before placing; searching unrelated surfaces before checking obvious object-holding surfaces.

Entry Written from an Expert-Assisted Trajectory (Two Objects)

title: Search dresser for second pencil

principle: When a task requires collecting two instances of a small handheld object and one has already been placed at the destination, search remaining unvisited surfaces, prioritizing dressers and drawers, before concluding the second instance is unavailable.

when_to_apply: Tasks requiring two identical small objects (e.g., pencils, pens) to be placed at a target location, after one instance has been found and placed.

procedure: 1. After placing the first object at the destination, note which surfaces have already been visited. 2. Search unvisited surfaces, prioritizing dressers and drawers, since small stationery items are commonly stored there. 3. Open closed containers before inspecting them. 4. If a surface yields the second instance, take it. 5. Return to the destination and place the second instance using the same move action syntax as the first.

preconditions: At least one instance of the target object has been found and placed at the destination; the second instance has not yet been located.

failure_modes: Skipping unvisited dressers or drawers and assuming no second instance exists; failing to open closed containers before inspecting; placing the second instance at the wrong location or using incorrect object name in the move action.