SKILLDUET: INTERNALIZING SKILLS THROUGH AGENT–ENVIRONMENT DUAL HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Skills are reusable strategies distilled from prior interactions that guide agents in task execution. Although skills improve inference-time performance, they do not become intrinsic agent capabilities, motivating skill internalization for autonomous execution without external guidance. However, existing methods typically rely on *start-of-trajectory* skill guidance, whose influence may gradually diminish over long interaction. We propose **SkillDuet**, a novel skill-internalization framework that can deliver *in-trajectory* guidance, providing persistent supervision at relevant decision points. To realize such supervision, the core of SkillDuet is agent–environment dual harnesses: beyond the start-of-trajectory guidance provided by an *agent harness*, an *environment harness* augments observations with skill-derived feedback to provide in-trajectory guidance. We further design a skill-internalization procedure, which collects paired assisted and unassisted trajectories, then converts successful assisted trajectories into skill-free supervision by removing external guidance. It internalizes skill-guided behavior through SFT with task-benefit weighting on converted trajectories, while using RL on unassisted trajectories to reinforce autonomous execution. Furthermore, to evaluate skill internalization in challenging long-horizon tasks, we introduce HARD-ALFWORLD, an ALFWorld variant with multiple goals, same-type distractors, and scene-grounded object references. Experiments on HARD-ALFWORLD and Search-QA with two backbones demonstrate that SkillDuet achieves leading performance in most settings and remains competitive across all evaluations.

1 INTRODUCTION

LLM agents have demonstrated strong capabilities in domains such as code generation (Yang et al., 2024), web browsing (Chae et al., 2025; Wei et al., 2025), and GUI interaction (Lin et al., 2025; Qin et al., 2025). Despite these advances, agent interactions are typically organized as isolated episodes, lacking a mechanism to accumulate experience from past trajectories for reuse in subsequent tasks (Liu et al., 2025; Wang et al., 2024). Such experience can be retained as episodic trajectory memory (Ferraz et al., 2026; Xiong et al., 2026), which preserves the details of individual interaction episodes, or abstracted into *skills*, which abstract away episode-specific details to capture more generalizable strategies for transfer across related tasks (Zhou et al., 2026; Shi et al., 2026b).

Alongside the emergence of agent harnesses (Chen et al., 2026b; Huang et al., 2026b), external skill banks (Wang et al., 2026b) have emerged as a promising mechanism for accumulating reusable skills. At inference time, relevant skills are typically retrieved from the bank as natural-language guidance (Huang et al., 2026a; Xia et al., 2026). Although effective, this paradigm incurs additional token overhead by injecting skills into the policy's input at inference time (Gao et al., 2026). It also requires continual maintenance of the external skill bank to keep stored skills non-redundant and useful, while reliable retrieval becomes increasingly challenging as the bank grows (Tan et al., 2026; Zhou et al., 2026). More importantly, external skills remain stored knowledge rather than becoming the agent's own capabilities, limiting the development of autonomous capabilities.

These limitations motivate skill internalization, which encodes reusable strategies into policy parameters for autonomous execution without external skill guidance at inference time (Lu et al., 2026). However, existing approaches (Shi et al., 2026a) typically rely on *start-of-trajectory guidance*, where skills are provided once with the task before the first decision. In long-horizon tasks,

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.

the influence of such guidance may weaken as the interaction history grows (He et al., 2024; Laban et al., 2026; Dongre et al., 2026), limiting its effect on later decisions and the guided behaviors available for learning. This reliance on initial guidance therefore limits persistent skill supervision throughout long-horizon execution.

To address these limitations, we propose **SkillDuet**, a skill-internalization framework built on *agent–environment dual harnesses*. The agent harness provides retrieved skills as start-of-trajectory guidance, while the environment harness compiles them into task-specific rules that deliver conditional feedback during interaction. Together, the two harnesses provide persistent skill guidance throughout long-horizon execution. To internalize this guidance, we design a procedure comprising paired trajectory collection, trajectory conversion, and policy optimization. It first collects paired assisted and unassisted trajectories, then converts successful assisted trajectories into skill-free supervision by removing external guidance while preserving the induced reasoning-action sequence. Policy optimization uses the *Paired Assistance Internalization and Reinforcement* (PAIR) objective, which applies task-weighted SFT to converted trajectories while using RL on unassisted trajectories. As the policy evolves, iterative skill discovery plus removal allows the skill bank to co-evolve with the policy. Our key contributions are summarized as follows:

- We propose SkillDuet, a skill-internalization framework built on agent—environment dual harnesses, which instantiate the same retrieved skills as start-of-trajectory guidance and conditional in-trajectory feedback, providing persistent supervision throughout long-horizon execution.
- We develop a skill-internalization procedure that integrates paired trajectory collection with skill-free conversion of successful assisted trajectories, then applies the PAIR objective to reproduce skill-guided behavior without external guidance.
- We construct HARD-ALFWORLD, a challenging ALFWorld variant with multiple instance-bound goals, same-type distractors, and initial-scene-grounded object references to evaluate skill internalization in long-horizon tasks. Experiments on HARD-ALFWORLD and Search-QA with two backbones consistently validate the effectiveness of SkillDuet across diverse evaluation settings.

2 RELATED WORK

2.1 AGENT–ENVIRONMENT INTERFACES AND HARNESSES

An *agent harness* is the runtime control layer surrounding an LLM policy that manages context, tools, state, control flow, and the agent–environment interface (Huang et al., 2026b; Chen et al., 2026b). SWE-agent (Yang et al., 2024) highlights the importance of designing agent-oriented action and observation interfaces, while subsequent work extends this idea by making such interfaces adaptive or learnable (Xu et al., 2026b; Wang et al., 2026c; Zhang et al., 2026a). Life-Harness (Xu et al., 2026b) turns recurring failures into reusable runtime interventions. EnvHarness (Huang et al., 2026a) diagnoses policy weaknesses and constructs programmable environment wrappers to support agent learning. HarnessBridge (Wang et al., 2026c) learns a bidirectional agent–environment interface that transforms observations and mediates actions before execution. These methods externalize experience into persistent interface modifications that remain part of the agent's execution loop. SkillDuet instead uses agent–environment dual harnesses only during training to provide persistent supervision for skill internalization, enabling autonomous execution in the original environment without external skill guidance at inference time.

2.2 SKILL INTERNALIZATION

Skills are reusable strategies distilled from prior interactions to guide task execution (Xia et al., 2026; Zhang et al., 2025). Existing work on skill discovery (Zhou et al., 2026; Yan et al., 2026), utilization (Tu et al., 2026; Tan et al., 2026), and selection (He et al., 2026a; Li et al., 2026a) typically stores such strategies in external skill banks for inference-time augmentation. *Skill internalization* instead encodes reusable strategies in policy parameters to enable autonomous execution, using policy distillation (Shi et al., 2026a; Xie et al., 2026), RL with explicit or implicit skill guidance (Lu et al., 2026; Zhang et al., 2026c; Wang et al., 2026d), or SFT on skill-guided trajectories, including those guided by structured skill graphs (Ding et al., 2026). Other work jointly studies skill utilization and internalization (Zhu et al., 2026; Lin et al., 2026) or selectively internalizes skills under predefined

criteria (Ji et al., 2026). Beyond the optimization objective, effective internalization also depends on how skill guidance shapes the behaviors available for learning. The influence of start-of-trajectory skill guidance may weaken as the interaction history grows in long-horizon tasks (He et al., 2024; Laban et al., 2026; Dongre et al., 2026), motivating persistent skill guidance at relevant decision points throughout execution. Skill0 (Lu et al., 2026) repeatedly supplies retained skills alongside compressed interaction history. SkillDuet instead compiles the retrieved skills with the current task into task-specific rule code, which provides step-level guidance. D2Skill (Tu et al., 2026) also provides step-level guidance, but retrieves step skills distilled from prior trajectories rather than deriving them from the same skills used for start-of-trajectory guidance. The two methods also differ in how assisted trajectories are used for learning. D2Skill applies RL while retaining external skill guidance in the assisted branch. SkillDuet instead removes both skill text and rule-based feedback from all successful assisted trajectories to create skill-free supervision. PAIR then uses SFT with task-benefit weighting on these converted trajectories to internalize skill-guided behaviors into the policy, while RL on unassisted trajectories reinforces autonomous execution.

3 PRELIMINARIES

Our method builds upon several standard components. We formulate agent–environment interaction for each task as a partially observable Markov decision process and adopt ReAct (Yao et al., 2022) as the agent execution paradigm. We also introduce the supervised fine-tuning (SFT) and reinforcement learning (RL) objectives used in our training objective.

3.1 PROBLEM SETUP

An *agent* is the system that interacts with an *environment* to complete a *task*. The task q specifies the problem and goals; the environment supplies the execution setting, including states, available actions, dynamics, observations, and rewards. The agent's *policy* π_θ is its LLM decision-making model, parameterized by θ. We denote the original environment configured for task q by $\langle \mathcal{X}, \mathcal{A}, \mathcal{O}, P, R_q, \rho_{0,q} \rangle$, where $\mathcal{X}$, $\mathcal{A}$, and $\mathcal{O}$ denote the state, action, and observation spaces, respectively; P is the transition function; R_q is the original terminal reward function; $\rho_{0,q}$ is the reset distribution. At interaction round t, the policy conditions on the trajectory prefix $\tau_t = (o_0, c_0, a_0, \ldots, c_{t-1}, a_{t-1}, o_t)$, where o_0 contains the task description and initial instructions. Following the ReAct paradigm, the policy generates reasoning c_t and then produces an action a_t, after which the environment returns the next observation o_{t+1}, yielding $\tau_{t+1} = \tau_t \oplus (c_t, a_t, o_{t+1})$. The environment verifier returns a binary terminal reward $R_q(\tau) \in \{0, 1\}$ indicating task success or failure. We further define $T = \mathrm{turns}(\tau)$ as the number of environment interaction rounds.

3.2 TRAINING ALGORITHMS

Existing optimization paradigms can be broadly categorized into SFT and RL (Guo et al., 2025). SFT learns from teacher trajectories through supervised imitation, whereas RL learns from environment rewards by reinforcing behaviors that lead to higher returns. We denote the complete policy output at interaction round t as $y_t = c_t \oplus a_t = (y_{t,1}, \ldots, y_{t,L_t})$, where $\oplus$ denotes sequence concatenation, $y_{t,k}$ is the k-th token generated at round t, and $L_t = |y_t|$ is the number of policy-generated tokens at this round. Each token is generated according to $\pi_\theta\left(y_{t,k} \mid q, \tau_t, y_{t,<k}\right)$, where $y_{t,<k} = (y_{t,1}, \ldots, y_{t,k-1})$ denotes the tokens generated earlier within the same interaction round.

Supervised Fine-Tuning. SFT trains the policy to imitate successful teacher trajectories $\mathcal{D}^{\mathrm{SFT}} = \{(q_i, \tau_i)\}_i$, where q_i denotes a task and τ_i denotes a successful teacher trajectory. The policy is optimized by minimizing the negative log-likelihood loss:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(q,\tau)\sim\mathcal{D}^{\mathrm{SFT}}}\left[\frac{1}{\sum_{t=0}^{T-1} L_t}\sum_{t=0}^{T-1}\sum_{k=1}^{L_t}\log \pi_\theta\left(y_{t,k} \mid q, \tau_t, y_{t,<k}\right)\right]. \tag{1}$$

Reinforcement Learning. At training step s, let $\mathcal{B}_s$ denote the batch of training tasks. For each task $q \in \mathcal{B}_s$, the agent interacts with the environment under policy π_θ to generate a trajectory group $\mathcal{G}_s(q)$, whose size may vary across tasks, yielding $\mathcal{D}_s^{\mathrm{RL}} = \bigcup_{q\in\mathcal{B}_s}\{(q,\tau) : \tau \in \mathcal{G}_s(q)\}$. For each $\tau^j \in \mathcal{G}_s(q)$, let $r_j = R_q(\tau^j)$ denote the environment reward, T_j the number of interaction

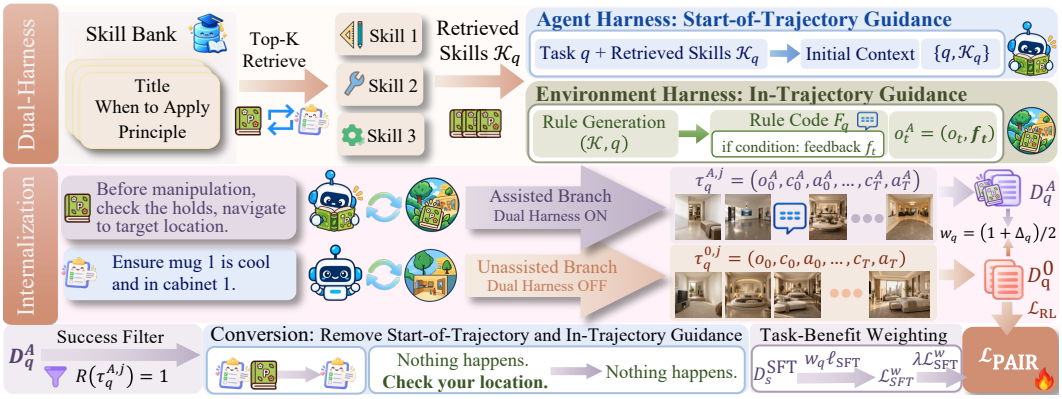


Figure 1: Overview of SkillDuet. Dual harnesses use the same retrieved skills for start-of-trajectory and conditional in-trajectory guidance. During internalization, assisted and unassisted trajectories are sampled. Successful assisted trajectories are converted into skill-free supervision, and PAIR combines task-benefit-weighted SFT on converted trajectories with RL on unassisted trajectories.

rounds, and $L_{j,t}$ the number of policy-generated tokens at round t. We define the clipped surrogate as $g_\epsilon(\rho, A) = \min\{\rho A, \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A\}$. Specifically, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and minimize

$$\mathcal{L}_{\text{RL}}\left(\theta; \mathcal{D}_s^{\text{RL}}\right) = -\frac{1}{|\mathcal{B}_s|} \sum_{q \in \mathcal{B}_s} \frac{1}{|\mathcal{G}_s(q)|} \sum_{j=1}^{|\mathcal{G}_s(q)|} \frac{1}{\sum_{t=0}^{T_j-1} L_{j,t}} \sum_{t=0}^{T_j-1} \sum_{k=1}^{L_{j,t}} g_\epsilon\left(\rho_{j,t,k}(\theta), \hat{A}_j\right), \tag{2}$$

where $\rho_{j,t,k}(\theta)$ is the token probability ratio of π_θ to the rollout policy $\pi_{\theta_{\text{old}}}$, $\hat{A}_j$ is the reward standardized within $\mathcal{G}_s(q)$ and shared across all policy-generated tokens in τ^j, and ϵ controls clipping.

4 Method

To enable skill internalization with persistent guidance throughout long-horizon interaction, SkillDuet comprises: *dual-harness skill guidance* and *skill internalization* (Fig. 1). The former provides start-of-trajectory guidance through retrieved skills and in-trajectory guidance through rule-based feedback compiled from the same retrieved skills, while the latter jointly optimizes SFT on converted skill-free trajectories and RL on unassisted trajectories under the PAIR objective to internalize skill-guided behaviors and reinforce autonomous execution. We further outline HARD-ALFWORLD construction for evaluating skill internalization in challenging long-horizon tasks.

4.1 Dual-Harness Skill Guidance

To provide persistent skill guidance, SkillDuet employs two complementary harnesses. The *agent harness* supplies the retrieved skills as start-of-trajectory guidance, while the *environment harness* transforms applicable skills into conditionally triggered feedback as in-trajectory guidance.

Agent Harness. At training step s, we retrieve a relevant skill set $\mathcal{K}_q$ for each task $q \in \mathcal{B}_s$ from the current skill bank $\mathcal{M}_s$ (Appendix B.2). In the assisted branch, the agent harness provides the full retrieved set $\mathcal{K}_q$ alongside the task instruction before the agent's first decision. For example, a HARD-ALFWORLD skill can be summarized as *title*: "Check manipulation prerequisites"; *when to apply*: "When a task requires cleaning an object"; and *principle*: "Pick up the object, confirm it is in your inventory, and go to the sink before cleaning it."

Environment Harness. The environment harness provides in-trajectory guidance by using rule code derived from retrieved skills to augment raw environment observations with conditionally triggered feedback. To generate this code, we prompt the current policy π_{θ_s} with task q and the retrieved skill set $\mathcal{K}_q$, using the rule-generation prompts in Appendices G.4 and G.7. The model produces a single block of task-specific rule code implementing the feedback function F_q. The code encodes

applicable strategies from the retrieved skills through multiple conditional branches, each specifying a triggering condition and the corresponding feedback. For example, if a cleaning action returns "Nothing happens.", a branch may emit: "Check your location with `look` and confirm you are holding the object with `inventory` before retrying." Rule conditions are evaluated using the task, interaction history, current action, and raw observation. Specifically, after the original environment executes action a_t and returns the raw observation o_{t+1}, the environment harness exposes the augmented observation to the policy:

$$o^A_{t+1} = (o_{t+1}, f_{t+1}), \qquad f_{t+1} = F_q(q, \tau_t, a_t, o_{t+1}),$$

where f_{t+1} is empty if none of its conditions is satisfied. Implementation details are provided in Appendix B.3; Appendix F illustrates the resulting guidance with trajectories and rule code.

4.2 Skill Internalization

To turn external skill guidance into autonomous capabilities, SkillDuet proceeds in two stages. The sampling stage performs *Paired Trajectory Collection* followed by *Trajectory Conversion*, while the training stage optimizes the policy using the PAIR objective.

Paired Trajectory Collection. At training step s, for each task with $\mathcal{K}_q \neq \emptyset$, we collect two sets of agent–environment trajectories using the same current policy π_{θ_s}: J unassisted trajectories in the task environment without external skill guidance, and J assisted trajectories in the same environment with both harnesses enabled. We denote the resulting trajectory sets by $\mathcal{D}^0_q = \{\tau^{0,j}_q\}_{j=1}^J$ and $\mathcal{D}^A_q = \{\tau^{A,j}_q\}_{j=1}^J$, where $\tau^{0,j}_q$ and $\tau^{A,j}_q$ denote the j-th unassisted and assisted trajectories for task q, respectively. If $\mathcal{K}_q = \emptyset$, we instead collect $2J$ unassisted trajectories and set $\mathcal{D}^A_q = \emptyset$, preserving the per-task sampling budget. In both cases, the RL trajectory group is $\mathcal{G}_s(q) = \mathcal{D}^0_q$.

Trajectory Conversion. To enable the policy to reproduce skill-guided behaviors without external guidance, we convert successful assisted trajectories for each task q into a skill-free set:

$$\widetilde{\mathcal{D}}^{A,+}_q = \left\{\widetilde{\tau} : \widetilde{\tau} = \Psi(\tau),\ \tau \in \mathcal{D}^A_q,\ R_q(\tau) = 1,\ \mathcal{K}_q \neq \emptyset\right\}, \tag{3}$$

where the deterministic operator Ψ removes the retrieved skill text supplied at the start of the trajectory and the rule-based feedback appended to observations. This conversion restores the original environment observations while preserving all policy-generated reasoning and actions induced by the guidance. Further conversion details are provided in Appendix B.4.

Policy Optimization with PAIR. The PAIR objective combines SFT on converted skill-free trajectories to internalize skills with RL on unassisted trajectories to reinforce autonomous execution. To prioritize supervision from tasks where assistance yields higher observed rewards, we estimate task-level assistance benefit by comparing the average rewards of the paired branches. For each task with paired trajectories, we derive an SFT weight from the difference in average reward between the assisted and unassisted branches:

$$\Delta_q = \frac{1}{J}\sum_{j=1}^{J}\left[R_q(\tau^{A,j}_q) - R_q(\tau^{0,j}_q)\right], \qquad w_q = \frac{1+\Delta_q}{2}. \tag{4}$$

Since $R_q(\tau) \in \{0,1\}$, we have $\Delta_q \in [-1,1]$ and $w_q \in [0,1]$. This *task-benefit weighting* assigns the same weight to all successful assisted trajectories for task q, emphasizing tasks with greater observed assistance benefit. Accordingly, the two training batches at step s are aggregated across tasks as

$$\mathcal{D}^{\mathrm{SFT}}_s = \bigcup_{q\in\mathcal{B}_s}\left\{(q,\widetilde{\tau},w_q) : \widetilde{\tau} \in \widetilde{\mathcal{D}}^{A,+}_q\right\}, \qquad \mathcal{D}^{\mathrm{RL}}_s = \bigcup_{q\in\mathcal{B}_s}\left\{(q,\tau) : \tau \in \mathcal{D}^0_q\right\}. \tag{5}$$

Let $\ell_{\mathrm{SFT}}(\theta; q, \widetilde{\tau})$ denote the per-trajectory negative log-likelihood averaged over policy-generated tokens, as in the SFT objective in Sec. 3. The weighted SFT loss is

$$\mathcal{L}^w_{\mathrm{SFT}}\left(\theta; \mathcal{D}^{\mathrm{SFT}}_s\right) = \frac{1}{|\mathcal{D}^{\mathrm{SFT}}_s|}\sum_{(q,\widetilde{\tau},w_q)\in\mathcal{D}^{\mathrm{SFT}}_s} w_q\,\ell_{\mathrm{SFT}}(\theta; q, \widetilde{\tau}). \tag{6}$$

We define $\mathcal{L}^w_{\mathrm{SFT}} = 0$ when $\mathcal{D}^{\mathrm{SFT}}_s = \emptyset$. At each training step, we optimize the policy with the RL objective on unassisted trajectories and the weighted SFT objective on skill-free trajectories:

$$\mathcal{L}_{\mathrm{PAIR},s}(\theta) = \mathcal{L}_{\mathrm{RL}}\left(\theta; \mathcal{D}^{\mathrm{RL}}_s\right) + \lambda_{\mathrm{SFT}}\mathcal{L}^w_{\mathrm{SFT}}\left(\theta; \mathcal{D}^{\mathrm{SFT}}_s\right), \tag{7}$$

Table 1: In-distribution performance comparison on HARD-ALFWORLD and Search-QA. SFT initialization for Qwen3-4B-Instruct-2507 applies only to HARD-ALFWORLD. **Bold** and underlined values indicate the best and second-best results in each performance column, respectively. † marks methods that use skills during evaluation. *Toks.* are reported in thousands.

Method	HARD-ALFWORLD valid-seen								Search-QA				
	Direct	Single	Dual	Triple	Quad	Toks.	Turns	Avg. Succ.	NQ	Triv	Toks.	Turns	Avg. EM
Qwen3-4B-Instruct-2507													
ReAct	5.9	3.6	2.4	0.0	0.0	61.8	31.7	2.4 ± 0.7	15.7	48.5	1.3	1.6	32.1 ± 0.9
GRPO	77.4	75.0	53.6	41.7	33.3	85.3	38.1	56.2 ± 0.7	37.1	60.0	2.1	2.7	48.5 ± 1.3
Skill0	67.8	71.4	54.8	**46.4**	27.4	81.1	20.9	53.6 ± 2.5	37.6	60.0	4.9	3.1	48.8 ± 0.3
D2Skill†	73.8	77.4	58.3	35.7	33.3	156.6	32.9	55.7 ± 2.1	**38.1**	60.8	2.1	2.5	$\mathbf{49.5 \pm 0.7}$
EnvHarness†	59.5	41.6	29.8	16.6	5.9	65.1	32.9	30.7 ± 1.0	28.8	55.5	2.4	2.2	42.1 ± 1.5
Skill0.5†	76.2	75.0	57.1	45.2	**39.3**	126.4	39.1	58.6 ± 1.2	32.0	53.1	5.6	2.8	42.5 ± 0.4
SkillDuet (Ours)	**78.6**	**82.1**	**59.5**	45.2	38.1	85.1	38.1	$\mathbf{60.7 \pm 1.5}$	37.1	**61.3**	2.7	2.6	49.2 ± 0.9
Qwen2.5-7B-Instruct													
ReAct	6.0	0.0	0.0	0.0	0.0	42.9	25.2	1.2 ± 0.7	7.5	34.7	1.7	2.5	21.1 ± 0.5
GRPO	67.9	35.7	34.5	**21.5**	**7.1**	81.7	34.6	33.3 ± 0.9	41.3	61.9	1.7	2.5	51.6 ± 1.4
Skill0	76.2	35.7	32.2	19.0	5.9	104.0	30.9	33.8 ± 0.7	44.5	60.5	3.4	2.3	52.5 ± 0.2
D2Skill†	70.2	34.5	32.1	15.5	6.0	99.4	25.2	31.7 ± 0.3	41.3	60.8	2.3	2.8	51.1 ± 1.0
EnvHarness†	14.3	0.0	2.4	0.0	0.0	54.4	28.0	3.3 ± 0.7	28.5	51.5	2.9	2.5	40.0 ± 1.0
Skill0.5†	**82.1**	48.8	27.4	19.0	1.2	101.5	32.7	35.7 ± 2.0	42.7	56.5	2.8	2.7	49.6 ± 0.3
SkillDuet (Ours)	64.3	**60.7**	**41.7**	17.9	**7.1**	101.5	43.0	$\mathbf{38.3 \pm 0.9}$	**45.1**	**62.7**	2.2	2.7	$\mathbf{53.9 \pm 1.0}$

where λ_{SFT} controls the global contribution of supervised internalization.

Overall, SkillDuet combines SFT on converted successful assisted trajectories with RL on unassisted trajectories to train the policy for autonomous execution without external skill guidance. This learning process integrates iterative skill discovery and removal to enable continual policy–skill co-evolution, with details provided in Appendix B.5 and complete training procedure in Appendix B.6.

4.3 HARD-ALFWORLD CONSTRUCTION

To evaluate skill internalization under more challenging long-horizon interactions, we construct HARD-ALFWORLD by extending ALFWorld (Shridhar et al., 2020) along three dimensions. **(1) Multiple goals.** Multiple goals create longer-horizon tasks that require agents to coordinate independent subgoals while tracking their completion throughout execution. We augment each task with two auxiliary goals, one for a same-type object and one for a different-type object. Both require placement in designated receptacles, and one additionally requires a state transformation. **(2) Same-type distractors.** Same-type distractors make object types alone insufficient for target selection, thereby requiring fine-grained object discrimination during execution. We introduce additional instances of target object types with varying initial states and locations, requiring agents to identify the correct objects from observable contextual cues rather than object types alone. **(3) Grounded references.** Grounded references require agents to identify target objects in the initial scene while preserving their identities despite subsequent state or location changes. We use two forms of object reference: numbered references based on object indices and descriptive references based on the initial scene. Each descriptive reference combines at most three observable conditions drawn from object state, co-location, and receptacle information to uniquely identify the target without redundancy. Appendix B.1 details task construction, validation, task grouping, and dataset statistics.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Training and evaluation settings. We conduct experiments on HARD-ALFWORLD, our challenging variant of ALFWorld (Shridhar et al., 2020), and Search-QA, a search-augmented QA suite following Search-R1 (Jin et al., 2025). For HARD-ALFWORLD, we train on the training split, with

Table 2: Out-of-distribution performance comparison on the HARD-ALFWORLD and Search-QA. SFT initialization for Qwen3-4B-Instruct-2507 applies only to HARD-ALFWORLD. **Bold** and underlined values indicate the best and second-best results in each performance column, respectively. † marks methods that use skills during evaluation. *Toks.* are reported in thousands.

Method	HARD-ALFWORLD valid-unseen								Search-QA							
	Direct	Single	Dual	Triple	Quad	Toks.	Turns	Avg. Succ.	Pop	Hotp	2Wk	MuS	Bam	Toks.	Turns	Avg. EM
Qwen3-4B-Instruct-2507																
ReAct	3.7	1.2	0.0	0.0	0.0	54.3	28.8	1.0 ± 0.4	25.3	28.5	18.4	7.2	32.0	2.2	2.2	22.3 ± 0.5
GRPO	75.3	70.4	49.4	**50.6**	20.5	87.6	39.1	53.5 ± 1.5	41.6	38.9	34.4	14.7	40.3	3.0	2.8	34.0 ± 0.5
Skill0	71.6	67.9	46.9	43.2	16.7	77.7	20.1	49.5 ± 0.9	**44.5**	**42.9**	35.7	**19.5**	33.3	5.6	3.5	35.2 ± 0.3
D2Skill†	69.1	70.3	40.7	45.7	**24.4**	162.3	33.8	50.2 ± 1.5	34.1	39.7	29.9	18.9	**43.2**	3.0	3.0	33.2 ± 0.4
EnvHarness†	49.4	42.0	32.1	9.9	3.8	58.6	30.8	27.6 ± 2.2	37.6	38.9	30.7	13.1	33.9	3.2	2.5	30.8 ± 1.3
Skill0.5†	72.9	**74.1**	46.9	49.4	16.7	135.5	41.2	52.3 ± 2.8	27.5	29.6	20.3	14.1	41.3	6.6	3.3	26.6 ± 0.9
SkillDuet (Ours)	**76.6**	71.6	**55.6**	**50.6**	19.2	86.9	39.1	$\mathbf{55.0 \pm 1.5}$	44.3	40.3	**37.1**	15.7	42.4	3.8	3.0	$\mathbf{35.9 \pm 0.7}$
Qwen2.5-7B-Instruct																
ReAct	3.7	0.0	0.0	1.2	0.0	89.3	39.0	1.0 ± 0.9	16.0	13.6	8.0	2.7	15.5	2.4	3.1	11.2 ± 0.3
GRPO	49.4	44.4	12.3	13.6	3.8	106.4	45.1	24.9 ± 1.3	39.7	38.4	38.4	15.5	38.4	2.7	3.2	34.1 ± 0.2
Skill0	51.9	51.8	**21.0**	9.9	3.8	117.1	34.0	27.8 ± 1.9	**47.2**	44.5	41.3	**22.7**	37.1	4.5	2.9	38.6 ± 0.5
D2Skill†	50.6	45.7	17.3	13.6	**6.4**	110.5	27.1	26.9 ± 1.2	42.9	41.1	33.9	20.0	34.4	3.0	3.2	34.5 ± 0.3
EnvHarness†	14.8	0.0	0.0	0.0	0.0	87.5	37.9	3.0 ± 0.0	41.9	36.8	28.3	15.2	32.8	4.0	3.0	31.0 ± 0.5
Skill0.5†	**55.5**	34.5	14.8	3.7	3.8	116.7	36.1	22.6 ± 0.3	45.6	42.4	42.4	16.5	37.3	3.6	3.1	36.9 ± 0.5
SkillDuet (Ours)	53.1	**55.6**	**21.0**	**17.3**	3.8	80.8	34.3	$\mathbf{30.3 \pm 2.1}$	44.8	**46.1**	**44.5**	21.1	**42.7**	3.1	3.3	$\mathbf{39.8 \pm 0.9}$

valid-seen used for in-distribution (ID) evaluation and valid-unseen for out-of-distribution (OOD) evaluation. Tasks are grouped by object-reference strategies into five groups: *Direct*, *Single*, *Dual*, *Triple*, and *Quad*, detailed in Appendix B.1. For Search-QA, we train on the NQ (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017). Their test sets are used for ID evaluation, while five additional datasets serve as OOD benchmarks. We use Qwen2.5-7B-Instruct (Qwen Team, 2025) and Qwen3-4B-Instruct-2507 (Yang et al., 2025). For HARD-ALFWORLD, all methods initialize Qwen3-4B-Instruct-2507 via SFT on the same 500 successful trajectories collected during GRPO training of Qwen2.5-7B-Instruct. We report three-run averages by task group or data source, along with overall averages, using success rate (*Avg. Succ.*), exact match (*Avg. EM*), tokens (*Toks.*, in thousands), and interaction rounds (*Turns*). See detailed settings in Appendices C.1 and C.2.

Baselines. We compare SkillDuet against ReAct, vanilla GRPO, and methods spanning different degrees of skill internalization. Skill0 (Lu et al., 2026) and D2Skill (Tu et al., 2026) use external skills to guide policy learning, while Skill0.5 (Zhu et al., 2026) internalizes generalizable skills but retains task-specific skills externally. We use EnvHarness (Huang et al., 2026a) as an external-skill inference augmentation baseline, with skill retrieval retained only at inference time. Method descriptions and implementation details are provided in Appendix C.3.

5.2 EXPERIMENTAL RESULTS

Performance. (1) As shown in Tables 1 and 2, SkillDuet achieves the best performance in most settings. The only exception is Search-QA with Qwen3-4B-Instruct-2507, where it remains highly competitive with D2Skill (49.2 vs. 49.5). These results demonstrate the strong effectiveness of our method. (2) To examine how tasks of different horizons contribute to SkillDuet's overall performance gains, we analyze SkillDuet's gains across task horizons by pooling the HARD-ALFWORLD valid-seen and valid-unseen splits, then grouping tasks by reference trajectory length as a proxy for task horizon. Fig. 2 decomposes SkillDuet's overall success-rate gains over GRPO, Skill0, Skill0.5, and D2Skill by reference trajectory length. Each group's contribution is computed as the difference in successful task counts divided by the total number of pooled evaluation tasks (274), expressed in percentage points; group sizes are provided in Appendix B.1.6. All methods are initialized from Qwen2.5-7B-Instruct. SkillDuet generally achieves positive gains on long-horizon

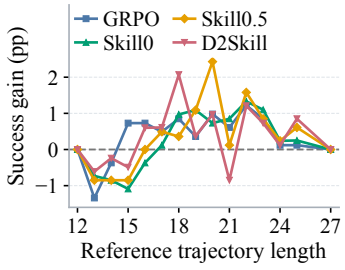
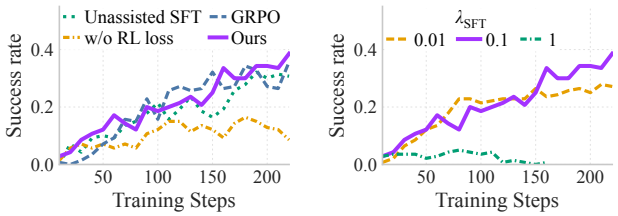


Figure 2: Contributions to overall success gains by reference trajectory length.

tasks (17–25 steps), supporting our motivation for combining start-of-trajectory guidance with in-trajectory feedback. On shorter tasks, the baselines are often stronger. One possible explanation for this horizon-dependent pattern is the allocation of the trajectory sampling budget between assisted and unassisted trajectories. On shorter tasks, unassisted exploration may already provide useful reward variation, so allocating fewer samples to RL could limit learning efficiency. On longer tasks, successful guided trajectories may provide complementary supervision when unassisted rewards are sparse, potentially helping the policy acquire behaviors that support subsequent RL.

Effects of dual-harness. Tab. 3 evaluates dual-harness skill guidance using Qwen2.5-7B-Instruct on HARD-ALFWORLD valid-seen. (1) *w/o agent harness* retains only conditionally triggered in-trajectory feedback, while *w/o environment harness* retains only start-of-trajectory guidance. Combining both harnesses yields a higher mean aggregate success rate than either variant in this setting, although the gains are not uniform across task groups. (2) *Triggered full-skill guidance* retains the agent harness and the triggering conditions of the environment harness but replaces rule-based feedback with the full retrieved skill text. Its lower mean aggregate success rate is consistent with a benefit from task-specific rule feedback over repeating the retrieved skills in this setting. (3) *Rule generation without skills* omits retrieved skills during rule generation. Its lower mean aggregate success rate suggests a benefit from conditioning rule generation on retrieved skills in this setting. Appendix G.5 details the prompt modifications used for this ablation.

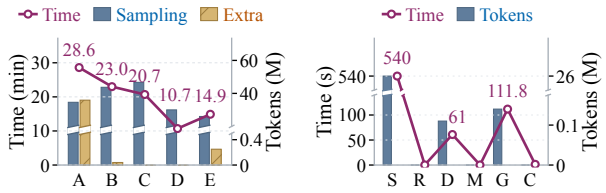
Effects of paired-trajectory internalization. Fig. 3(a) compares variants of the PAIR objective using Qwen2.5-7B-Instruct on HARD-ALFWORLD valid-seen. (1) *Unassisted SFT* retains RL on unassisted trajectories but replaces converted assisted trajectories with successful unassisted ones for SFT. The full method achieves higher final aggregate success under the evaluated configurations. Differences in successful SFT trajectory counts, token counts, or effective SFT weights may also contribute to this gap, so this comparison alone does not isolate the effect of skill-guided content. (2) The full method also achieves higher final aggregate success than GRPO in this setting. (3) *w/o RL loss* applies SFT only to converted skill-free trajectories. Its lower final aggregate success rate suggests a benefit from combining imitation with RL in this setting. (4) In Fig. 3(b), $\lambda_{\mathrm{SFT}} = 0.1$ achieves the highest final aggregate success rate among the tested values. Both 0.01 and 1 yield lower final success rates, indicating sensitivity to the relative weight of SFT and RL in this setting. Additional analyses of task-benefit weighting and trajectory conversion are provided in Appendix D.



(a) Training dynamics. (b) SFT coefficient λ_{SFT}.

Figure 3: Effects of paired-trajectory internalization on HARD-ALFWORLD valid-seen. (a) Success rates over training steps across different method settings. (b) Success rates under different λ_{SFT} values in the PAIR objective.

Training Efficiency. As shown in Fig. 4, SkillDuet introduces several additional operations beyond vanilla GRPO. (1) Most of the additional overhead comes from skill discovery and rule generation, both of which require additional model calls. Rule generation uses the current policy to translate retrieved skills into rule code for the environment harness, while skill discovery distills new skills from unassisted trajectories to update the skill bank. These operations add only approximately 1.0% to the token cost of trajectory sampling and 19.3% to the total per-step training time. (2) We further compare the per-step training time across baselines. Except for vanilla GRPO, SkillDuet has the lowest training cost per step,



(a) Costs per training step. (b) Costs by training operation.

Figure 4: Training efficiency of Qwen2.5-7B-Instruct, measured by average time and token cost per training step. (a) Methods: *A*: Skill0.5; *B*: D2Skill; *C*: Skill0; *D*: GRPO; *E*: Ours. (b) Operations of Ours: *S*: trajectory sampling; *R*: skill retrieval; *D*: skill discovery; *M*: skill removal; *G*: rule generation; *C*: trajectory conversion.

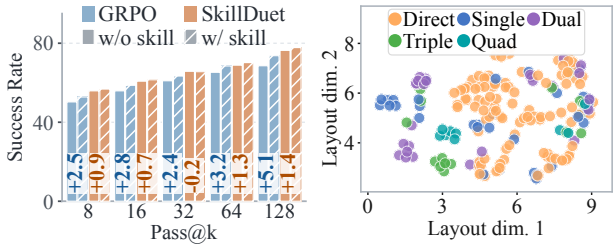
Table 3: Ablation of dual-harness using Qwen2.5-7B-Instruct. *Toks.* are reported in thousands.

Variant	HARD-ALFWORLD valid-seen							
	Direct	Single	Dual	Triple	Quad	Toks.	Turns	Avg. Succ.
Triggered full-skill guidance	65.5	35.7	39.3	13.1	10.7	122.5	44.7	32.8 ± 1.0
Rule generation without skills	69.1	36.9	32.1	15.5	7.1	90.8	37.4	32.1 ± 1.8
SkillDuet (Ours)	64.3	60.7	41.7	17.9	7.1	101.5	43.0	38.3 ± 0.9
w/o environment harness	63.1	38.1	32.2	19.1	9.5	146.9	41.9	32.4 ± 0.7
w/o agent harness	65.5	35.7	39.3	16.6	11.9	127.2	44.1	33.8 ± 0.3

requiring only 39.3% more time than GRPO. We consider this moderate overhead acceptable given the additional skill-guided training mechanisms.

5.3 ANALYSIS OF SKILL GUIDANCE AND SKILL DIVERSITY

Skill Guidance Scaling. To assess whether SkillDuet reduces reliance on external skill guidance and how this behavior changes across sampling budgets, we compare GRPO- and SkillDuet-trained policies initialized from Qwen2.5-7B-Instruct under both guided and unguided evaluation. We use the skills discovered during SkillDuet training as a shared external skill bank, ensuring that both policies receive guidance from the same source. As shown in Fig. 5(a), external skill guidance generally improves both policies across the evaluated sampling budgets, but yields smaller gains for the SkillDuet-trained policy than for the GRPO-trained policy. Because the guidance source is held fixed, this reduced marginal benefit is consistent with SkillDuet relying less on external skills. Meanwhile, SkillDuet consistently outperforms GRPO under both guided and unguided evaluation, with the advantage persisting across sampling budgets. Together, these results support the effectiveness of SkillDuet and provide indirect evidence that part of the externally provided skill knowledge has been internalized.



(a) Skill guidance scaling. (b) Skill bank distribution.

Figure 5: Skill guidance and skill bank analysis. (a) Pass@k on HARD-ALFWORLD valid-seen across sampling budgets, with and without skill. (b) Distribution of skills discovered during SkillDuet training.

Skill Bank Distribution. To analyze the semantic diversity of discovered skills, we embed the skills and project their representations using UMAP (McInnes et al., 2018). As shown in Fig. 5(b), the resulting distribution exhibits both cross-group and within-group clustering. Cross-group clusters often capture task-agnostic principles, such as *Order Independent Subgoals to Minimize Travel*. Such principles concern how an agent organizes its actions and can apply across tasks with different goals. In contrast, task-group-specific strategies tend to cluster within their respective task groups, reflecting shared requirements and recurring interaction patterns. These patterns suggest that the skill bank contains generalizable behavioral knowledge at different levels of scope, ranging from principles shared across task groups to strategies reusable within particular task groups.

6 CONCLUSION

We introduced SkillDuet, a framework for internalizing reusable skills through persistent guidance in long-horizon interactions. By combining dual-harness guidance with skill internalization, SkillDuet learns from assisted and unassisted trajectories to improve autonomous task execution, while an evolving skill bank supports continual policy–skill co-evolution. Across Search-QA and HARD-ALFWORLD, SkillDuet achieves the best mean aggregate score in seven of eight settings and consistently outperforms GRPO. Despite its effectiveness, our framework has been evaluated only on text-only benchmarks. Extending the evaluation to multimodal benchmarks remains an important direction for future work.

AI USE STATEMENT

In this work, we used AI-assisted development tools to support the implementation of our proposed method. We also used AI tools to improve the readability and presentation of the manuscript and to check compliance with submission formatting requirements. We did not use generative AI tools to generate datasets, develop theoretical or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design the research methodology or experiments, or interpret experimental results.

All AI-assisted code was reviewed and tested by the authors for correctness, and all AI-assisted text was carefully reviewed and revised by the authors. Submission requirements were independently verified against the official conference guidelines. We take full responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work introduces SkillDuet, a framework that enables LLM agents to internalize external skill guidance for autonomous task execution. Our experiments focus on simulated embodied tasks in HARD-ALFWORLD and question-answering tasks in Search-QA. While advances in agent capabilities may have broader societal implications, we do not foresee ethical concerns specific to the methods or experiments presented in this work that would conflict with the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we describe SkillDuet in Section 4, with additional method details provided in Appendix B. The complete training procedure and the construction of HARD-ALFWORLD are also included in Appendix B. Appendix C documents the hardware, training data, evaluation data, hyperparameters, protocols, and baseline implementations. Prompt templates for agent execution, skill discovery, and rule generation are provided in Appendix G. Our implementation and the HARD-ALFWORLD dataset are publicly available at `https://anonymous.4open.science/r/verl_skill_iclr-4543`.

REFERENCES

Parul Awasthy, Aashka Trivedi, et al. Granite embedding r2 models. *arXiv preprint arXiv:2508.21085*, 2025.

Jaime G Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *ICLR*, 2025.

Haolong Chen, Liang Zhang, et al. Erskill: Evolving for skill-guided adaptive memory retrieval. *arXiv preprint arXiv:2608.12720*, 2026a.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026b.

Fei Ding, Yongkang Zhang, Runhao Liu, Yuhao Liao, Zijian Zeng, and Huiming Yang. Scaffold-mediated post-training: Co-evolving model parameters and procedural scaffold graphs. *arXiv preprint arXiv:2608.05156*, 2026.

Vardhan Dongre, Joseph Hsieh, Viet Dac Lai, Seunghyun Yoon, Trung Bui, and Dilek Hakkani-Tür. When attention closes: How llms lose the thread in multi-turn interaction. *arXiv preprint arXiv:2605.12922*, 2026.

Thomas Palmeira Ferraz, Romain Deffayet, Vassilina Nikoulina, Hervé Déjean, and Stéphane Clinchant. Retrieval-augmented llm agents: Learning to learn from experience. *arXiv preprint arXiv:2603.18272*, 2026.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Liang He, Jingbo Wen, Hongyu Gu, Hao Li, Haoyu Wang, Yixiong Chen, Kangning Cui, and Xilu Wang. From relevance to execution utility: Reward-aware dynamic execution gating for skill-based llm agents. *arXiv preprint arXiv:2608.09168*, 2026a.

Yun He, Di Jin, Chaoqi Wang, Chloe Bi, Karishma Mandyam, Hejia Zhang, Chen Zhu, Ning Li, Tengyu Xu, Hongjiang Lv, et al. Multi-if: Benchmarking llms on multi-turn and multilingual instructions following. *arXiv preprint arXiv:2410.15553*, 2024.

Zhongyu He, Yuanfan Li, Fei Huang, Tianyu Chen, Siyuan Chen, Xingyang Li, Meng Hsuan Yu, Xiangrong Liu, Leyi Wei, Lu Pan, et al. Siri: Self-internalizing reinforcement learning with intrinsic skills for llm agent training. *arXiv preprint arXiv:2606.02355*, 2026b.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *COLING*, 2020.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, et al. Envharness: Awakening static worlds for agent learning. *arXiv preprint arXiv:2608.19880*, 2026a.

Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, Yue Zhao, and Xiangliang Zhang. Memoharness: Agent harnesses that learn from experience. *arXiv preprint arXiv:2607.14159*, 2026b.

Tianyun Ji, Zhenya Huang, Jiayu Liu, Zirui Liu, Yu Su, and Hongbin Pei. Learning what to remember and what to internalize in llm self-evolution via adaptive memory-parameter coordination. *arXiv preprint arXiv:2608.01234*, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*, 2017.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *TACL*, 2019.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *ICLR*, 2026.

Qinfeng Li, Dalin He, Yuntai Bao, Ying Yang, Ruoxi Chen, Xinyan Yu, Lizhou Liang, Ge Su, Wenqi Zhang, and Xuhong Zhang. Skillaligner: Treating retrieved skills as adaptable drafts at execution time. *arXiv preprint arXiv:2608.06880*, 2026a.

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

Huizu Lin, Chengkai Huang, Tianqi Gao, Tao Huang, Daijiao Liu, Tongxin Li, Xiaoyan Sun, and Lina Yao. Auso: Action-level unified skill optimization from internalization to utilization. *arXiv preprint arXiv:2608.21292*, 2026.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *CVPR*, 2025.

Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *ACL*, 2025.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *ACL*, 2023.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Dilxat Muhtar, Jiashun Liu, Wei Gao, Weixun Wang, Shaopan Xiong, Ju Huang, Siran Yang, Wenbo Su, Jiamang Wang, Ling Pan, et al. Complementary reinforcement learning. *arXiv preprint arXiv:2603.17621*, 2026.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of EMNLP*, 2023.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Shijie Ren, Xiting Wang, Meng Li, Yujie Guo, Yunhang Yao, Ziheng Peng, Xunlong Wang, Yuetan Chen, Haoyang Zhou, Yunlong Liang, et al. Self-improving large language models via progressive experience evolution. *arXiv preprint arXiv:2608.02139*, 2026.

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

Tianyuan Shi, Canbin Huang, Bei Li, Xin Chen, Xiaojun Quan, Jingang Wang, and Qifan Wang. Beyond trajectory imitation: Strategy-guided policy optimization for llm reasoning. *arXiv preprint arXiv:2606.24064*, 2026a.

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*, 2026b.

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

Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, Liming Zhu, and Wenjie Zhang. Skillzip: Contract-preserving graph compression for scalable agent skill libraries. *arXiv preprint arXiv:2608.05604*, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *TACL*, 2022.

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.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026a.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *ACL*, 2026b.

Xiaoxuan Wang, Haixin Wang, Alexander Taylor, Jason Cong, Yizhou Sun, and Wei Wang. Harness-bridge: Learnable bidirectional controller for llm agent harness. *arXiv preprint arXiv:2606.12882*, 2026c.

Zi-Han Wang, Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Jie Wu, Zhengzhou Cai, Yueqing Sun, Ziang Ye, Linji Hao, Qi Gu, et al. Agentopsd: Recursive self-distillation for agentic reinforcement learning. *arXiv preprint arXiv:2608.05987*, 2026d.

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

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, et al. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *EMNLP*, 2025.

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

Can Xie, Yuyi Zhou, Wen Yang, Ziyi zhang, Siyao Song, Yingzhuo Deng, Shuo Ren, and Jiajun Zhang. Edge: Experience-distillation for guided exploration in agentic reinforcement learning. *arXiv preprint arXiv:2608.21946*, 2026.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. In *ACL*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic llm agents. *arXiv preprint arXiv:2605.22166*, 2026b.

Qianxi Yan, Chunrong Chen, Jiuzhou Zhao, Min Zhang, Yongzhou Xu, and Xiaochuan Xu. Skillevo: Self-renewing evolution gradients from multi-turn interaction feedback. *arXiv preprint arXiv:2608.13120*, 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.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *NeurIPS*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. `https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills`, 2025.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, et al. Jit-agent: Scaling harness intelligence via just-in-time harness evolution. *arXiv preprint arXiv:2608.25593*, 2026a.

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

Ranxu Zhang, Guinan Chen, et al. Agentic reinforcement learning with self-distilled reward shaping. *arXiv preprint arXiv:2608.03223*, 2026c.

Xiaoying Zhang, Zichen Liu, Yipeng Zhang, Xia Hu, and Wenqi Shao. Retroagent: From solving to evolving via retrospective dual intrinsic feedback. *arXiv preprint arXiv:2603.08561*, 2026d.

Zhiwei Zhang, Yudi Lin, Nikki Lijing Kuang, Linlin Wu, Xiaomin Li, Songtao Liu, and Fenglong Ma. Co-evolving skill generation and policy optimization. *arXiv preprint arXiv:2606.08755*, 2026e.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, et al. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

Jiapeng Zhu, Jianxiang Yu, Yibo Zhao, Chengcheng Han, Qi Gu, Xunliang Cai, Xiang Li, and Weining Qian. Skill0.5: Joint skill internalization and utilization for out-of-distribution generalization in agentic reinforcement learning. *arXiv preprint arXiv:2605.28424*, 2026.

TABLE OF CONTENTS

A Additional Related Works **16**

A.1 Skill and Model Co-evolution 16

B Additional Method Details **16**

B.1 Hard-ALFWorld Construction 16

B.2 Skill Retrieval 19

B.3 Environment Harness 20

B.4 Trajectory Conversion 20

B.5 Dynamic Skill-Bank Maintenance 20

B.6 Complete Training Procedure 22

C Implementation Details **22**

C.1 Training Settings 22

C.2 Evaluation Settings 24

C.3 Baseline Methods and Implementation 25

D Additional Analysis of Skill Internalization **26**

E Training Stability across Random Seeds **27**

F Case Study **28**

F.1 Rule Code Excerpts 30

F.2 Trajectory Conversion 31

G Prompt Templates **32**

G.1 Prompts for HARD-ALFWORLD Agent Execution 32

G.2 Prompts for Search-QA Agent Execution 34

G.3 Prompts for HARD-ALFWORLD Skill Discovery 35

G.4 Prompt of HARD-ALFWORLD Rule Generation 43

G.5 Prompt Modifications for Rule Generation without Skills 51

G.6 Prompts for Search-QA Skill Discovery 56

G.7 Prompt of Search-QA Rule Generation 65

A Additional Related Works

A.1 Skill and Model Co-evolution

Skill–model co-evolution extends skill learning by updating the external skill bank alongside the policy or an auxiliary model. For *policy–skill co-evolution*, one line of work emphasizes skill utilization, continually generating or refining skills while training the policy to use them for task execution (Zhang et al., 2026e; Shi et al., 2026b; Li et al., 2026b; Zhang et al., 2026d). Another emphasizes skill internalization, progressively encoding newly acquired strategies in policy parameters to reduce reliance on external skill guidance (Ren et al., 2026; Wang et al., 2026a; He et al., 2026b). Related work also co-evolves skills with verifiers that test and refine them (Zhang et al., 2026b) or with skill extractors that adapt to the evolving policy (Muhtar et al., 2026). ERSkill (Chen et al., 2026a) co-evolves executable memory-retrieval skills with a query-conditioned skill router, retaining the evolved skills for inference-time evidence construction. SkillDuet follows the internalization paradigm, integrating iterative skill discovery and removal with policy optimization under the PAIR objective. The current policy proposes new skills from selected unassisted trajectories, while paired assisted and unassisted rewards inform the removal of skills repeatedly associated with negative assistance benefit. These updates allow the skill bank to evolve with the policy, supplying reusable strategies for subsequent dual-harness guidance and internalization.

B Additional Method Details

This section supplements the main method with the construction of Hard-ALFWorld, skill retrieval, environment harness implementation, trajectory conversion, dynamic skill-bank maintenance, and the complete training procedure.

B.1 Hard-ALFWorld Construction

Hard-ALFWorld extends ALFWorld with multiple goals, same-type distractors, and object references grounded in the initial scene. It preserves the original action space, environment dynamics, and binary terminal rewards. Together, these extensions require agents to coordinate subgoals, identify the intended object instances, and track those instances through subsequent state or location changes. Construction proceeds in five stages. First, we construct tasks with multiple goals by binding the original goal to specific object instances and adding auxiliary goals. Second, we introduce same-type distractors to require discrimination between object instances. Third, we generate grounded references, configuring initial-state contrasts as needed to support the required reasoning patterns. Fourth, we generate task instructions, retaining only tasks that pass structural checks and trajectory replay. Finally, we classify the validated tasks by their reference forms and reasoning patterns while preserving ALFWorld's original dataset splits.

B.1.1 Multi-Goal Task Construction

Main-goal binding. We first parse each original scene and its reference trajectory to identify the main target objects, destination receptacles, and relevant lamps. We bind the main goal's success conditions to the specific PDDL object instances identified from the reference trajectory, rather than allowing any object of the same type to satisfy them.

Auxiliary goals. Starting from this instance-bound main goal, we add two auxiliary goals by default: one for an object of the same type as the main target and one for an object of a different type. Both require placement in designated receptacles, and one additionally requires cleaning, heating, cooling, or slicing. For placement-based tasks, the resulting goal is

$$\Omega_{\text{hard}} = \Omega_{\text{main}}^{\text{instance}} \wedge \text{In}(\xi_1, \zeta_1) \wedge \text{In}(\xi_2, \zeta_2) \wedge \text{State}(\xi_\ell), \qquad \ell \in \{1, 2\},$$

where $\Omega_{\text{main}}^{\text{instance}}$ is the instance-bound main goal, ξ_1, ξ_2 are the auxiliary object instances, and ζ_1, ζ_2 are their respective destination receptacles. The predicate In denotes placement in or on a receptacle, and $\text{State}(\xi_\ell)$ specifies the required state transformation for the selected auxiliary object.

Feasibility constraints. Auxiliary goals must be feasible and spatially separated. Each auxiliary object starts away from its destination. The two auxiliary goals must have distinct destination receptacles; neither receptacle coincides with the main destination. A state transformation is feasible only if the selected object supports it and the required appliance or tool is available. To meet these constraints, we add objects or facilities as needed, including different-type food objects, knives, and cleaning, heating, or cooling appliances. New facilities are associated with existing scene locations. We discard a scene only if no valid augmented task can be constructed after these additions.

B.1.2 Same-Type Distractor Introduction

We augment the multi-goal scene with same-type distractors so that object type alone is insufficient for target identification. We start with a base budget of two distractors allocated across the target object types. We add further instances as needed to cover every target type and satisfy the pattern-specific contrast requirements described below. The fully augmented scene contains at least two instances of each target object type. Distractors share their target's object type and supported operations, potentially also matching the initial cleanliness, temperature, or slicing state used to describe it. This construction applies to both numbered and descriptive references.

B.1.3 Grounded Reference Generation

We generate two forms of grounded reference: *numbered references* based on object indices and *descriptive references* based on initial states or relations. Descriptive references can involve four reasoning patterns: *conjunctive reasoning*, *co-location reasoning*, *multi-state reasoning*, and *identity tracking*. These patterns may overlap within a single reference. Destination receptacles and lamps retain their original ALFWorld indices under both reference forms.

Numbered references. Numbered references identify target instances directly by their ALFWorld object indices. Tasks using this form retain the augmented goals and same-type distractors.

Descriptive references. Descriptive references identify targets using conditions that hold at the start of the task and are observable through the existing environment interface. Object-state conditions cover clean, hot, cold, or sliced states. Contextual conditions cover the receptacle type, its open state, or co-location with an object of a specified type. We annotate each descriptive reference with the applicable patterns defined below. **(1) Conjunctive reasoning (`conjunctive`).** This pattern combines at least two conditions, including at least one condition other than an object state. For example, a reference may identify an apple as initially dirty and inside a closed cabinet. **(2) Co-location reasoning (`co_location`).** This pattern identifies a target through its relation to another object in the initial scene. We use a condition specifying that the target shares a receptacle or surface with an object of a given type, such as an apple initially sharing a receptacle with a knife. **(3) Multi-state reasoning (`multi_state`).** This pattern combines at least two object-state conditions, including at least one negation. To require a combination of negative temperature conditions, we select a suitable target that is initially neither hot nor cold, such as a tomato. We add one hot and one cold same-type distractor to its initial receptacle. Both distractors match the target's other attributes, such as cleanliness or sliced state. The condition "not hot" still matches the cold distractor; "not cold" still matches the hot one. Both conditions are therefore needed to identify the target. We add at most one such contrast group per scene. **(4) Identity tracking (`identity_tracking`).** This pattern uses an initial identifying attribute that changes under a required state transformation. To construct temperature-based cases, we select targets with observable temperature that support both heating and cooling. A target required to be heated starts cold, whereas one required to be cooled starts hot. A reference to the initially cold apple, for example, continues to identify that same apple after it has been heated. The agent must therefore preserve the target's identity even after the descriptive condition ceases to hold.

We select descriptive references based on the final augmented initial scene, accounting for all target objects and distractors. Each reference must uniquely identify its target using at most three conditions, each necessary for disambiguation. For target ξ^*, let $\mathcal{V}$ be its condition set and $\mathrm{Match}(\mathcal{V})$ the same-type objects satisfying all conditions in that scene. We require

$$|\mathcal{V}| \le 3, \qquad \mathrm{Match}(\mathcal{V}) = \{\xi^*\}, \qquad \forall \nu \in \mathcal{V}, \quad |\,\mathrm{Match}(\mathcal{V} \setminus \{\nu\})| > 1.$$

Table 4: Reasoning-pattern combinations within each HARD-ALFWORLD task group. C: conjunctive; L: co-location; M: multi-state; I: identity tracking. None denotes tasks using only numbered references.

Task group	Pattern combination	train ($N = 3{,}553$)	valid-seen ($N = 140$)	valid-unseen ($N = 134$)
Direct	None	711 (20.01%)	28 (20.00%)	27 (20.15%)
Single	C	1 (0.03%)	0 (0.00%)	0 (0.00%)
	L	650 (18.29%)	25 (17.86%)	24 (17.91%)
	M	3 (0.08%)	0 (0.00%)	0 (0.00%)
	I	57 (1.60%)	3 (2.14%)	3 (2.24%)
Dual	C + L	211 (5.94%)	9 (6.43%)	13 (9.70%)
	C + M	95 (2.67%)	2 (1.43%)	0 (0.00%)
	C + I	52 (1.46%)	2 (1.43%)	0 (0.00%)
	L + M	53 (1.49%)	2 (1.43%)	6 (4.48%)
	L + I	259 (7.29%)	12 (8.57%)	8 (5.97%)
	M + I	40 (1.13%)	1 (0.71%)	0 (0.00%)
Triple	C + L + M	260 (7.32%)	11 (7.86%)	8 (5.97%)
	C + L + I	276 (7.77%)	14 (10.00%)	15 (11.19%)
	C + M + I	112 (3.15%)	0 (0.00%)	0 (0.00%)
	L + M + I	63 (1.77%)	3 (2.14%)	4 (2.99%)
Quad	C + L + M + I	710 (19.98%)	28 (20.00%)	26 (19.40%)

Thus, removing any condition makes the reference ambiguous, ensuring that every included condition contributes to target identification.

B.1.4 INSTRUCTION GENERATION AND TASK VALIDATION

We combine the augmented goal with its object references to construct a final instruction specifying all goal states that must hold at task completion. The instruction explicitly states that object descriptions refer to the start of the task. The binary terminal reward is 1 if all augmented goal conditions are satisfied, and 0 otherwise.

Before retaining a task, we verify both structural consistency and solvability. Structural checks ensure that each object reference uniquely identifies its target in the complete initial scene, every descriptive condition is necessary for disambiguation, and all success conditions are bound to the intended instances. To verify solvability, we replay a reconstructed reference trajectory for the final target instances in the environment and confirm that it satisfies the augmented goal. Only tasks passing both structural validation and trajectory replay are retained. This procedure verifies that the reference trajectory completes the task; it does not establish shortest-path or optimality guarantees.

B.1.5 TASK CLASSIFICATION AND DATASET ORGANIZATION

Task groups. We organize the validated tasks into five groups based on their grounded references (Tab. 5). In *Direct Reference* (**Direct**), every target uses a numbered reference. For tasks using descriptive references, we take the union of reasoning-pattern labels across all object references. The number of distinct patterns determines the group: *Single-Pattern Reasoning* (**Single**), *Dual-Pattern Reasoning* (**Dual**), *Triple-Pattern Reasoning* (**Triple**), or *Quadruple-Pattern Reasoning* (**Quad**). These groups contain one through four distinct patterns, respectively. The same groups organize group-specific skills and evaluation results.

Tab. 4 reports the frequency of each exact task-level pattern combination within these groups. Rows are mutually exclusive because each task contributes to exactly one pattern set. Each entry gives a count with its percentage of the corresponding split's total tasks.

Dataset splits. We construct tasks independently within ALFWorld's training, valid-seen, and valid-unseen splits, preserving the original split boundaries. Each split is approximately balanced

Table 5: Task groups in HARD-ALFWORLD. A task's pattern set is the union of reasoning pattern labels across all object references. Combination counts enumerate subsets of the four reasoning patterns; they do not imply equal frequencies within a group.

Task group	Definition	Combinations
Direct	All targets are identified by numbered references; none of the four reasoning patterns is used.	None
Single	Exactly one distinct reasoning pattern occurs in the task.	4
Dual	Exactly two distinct reasoning patterns occur in the task.	$\binom{4}{2} = 6$
Triple	Exactly three distinct reasoning patterns occur in the task.	$\binom{4}{3} = 4$
Quad	All four reasoning patterns occur in the task.	1

Table 6: HARD-ALFWORLD split sizes and task group counts.

Split	Total	Direct	Single	Dual	Triple	Quad
train	3,553	711	711	710	711	710
valid-seen	140	28	28	28	28	28
valid-unseen	134	27	27	27	27	26

across the five task groups. All compared methods use the same generated instances. Tab. 6 reports the resulting split sizes and group counts.

Construction example. The following example illustrates how an original ALFWorld task is extended with auxiliary goals and grounded object references.

Original task.
heat some apple and put it in countertop.

Augmented task.
Your task is to: Ensure all of these final conditions hold. Object descriptions refer to the state at the start of the task. The apple that was clean and cold and in the same container or on the same surface as another apple is hot and in countertop 2. The apple that was not hot and not cold and in the same container or on the same surface as a bowl is in garbagecan 1. Pan 2 is cool and in cabinet 5. Use only the specified objects and numbered containers or lamps.

The augmented task retains the main goal of heating an apple, now bound to the instance described by its initial attributes. It adds two auxiliary goals: placing another apple in garbagecan 1 and ensuring pan 2 is cool in cabinet 5. Both apple descriptions combine state conditions with co-location, invoking conjunctive and co-location reasoning. The first also requires identity tracking as the initially cold apple becomes hot. The second requires multi-state reasoning through the conjunction of "not hot" and "not cold".

B.1.6 REFERENCE TRAJECTORY LENGTH DISTRIBUTION

For the long-horizon analysis in Section 5.2, we pool the 140 valid-seen and 134 valid-unseen tasks, yielding 274 evaluation tasks. We define reference trajectory length as the number of actions in the reconstructed reference trajectory whose successful replay is verified during task construction. We use this length as a proxy for task horizon, group tasks by this length, and compare methods on the same tasks within each group. Tab. 7 lists the number of distinct tasks at each observed length; repeated evaluation runs do not increase these counts. The distribution is concentrated around 16–23 steps, with fewer tasks at the extremes. In particular, the 12-step and 27-step groups contain only two tasks and one task, respectively, so comparisons at these lengths rest on few instances.

B.2 SKILL RETRIEVAL

At training step s, we retrieve a relevant skill set $\mathcal{K}_q$ from the current skill bank $\mathcal{M}_s$ for each task $q \in \mathcal{B}_s$. We use granite-embedding-english-r2 (Awasthy et al., 2025) as the embedding encoder E_ϕ. Let

Table 7: Task counts by reference trajectory length in the pooled HARD-ALFWORLD valid-seen and valid-unseen splits used in Fig. 2.

Reference trajectory length (steps)	12	13	14	15	16	17	18	19	20	21	22	23	24	25	27	**Total**
Tasks	2	12	14	15	28	27	42	21	35	32	16	14	6	9	1	**274**

T_q denote the task-description serialization template, while T_m denotes the skill-text serialization template. We encode the task description and the text of each skill in the bank as

$$e_q = E_\phi(T_q(q)), \qquad e_m = E_\phi(T_m(m)). \tag{8}$$

The skill bank is represented as

$$\mathcal{M}_s = \{(m, e_m, u_{m,s})\}, \tag{9}$$

where $u_{m,s}$ is the consecutive negative-utility counter associated with skill m.

We first retain skills whose embedding cosine similarity to the task description exceeds the retrieval threshold δ_{ret}:

$$\mathcal{C}_q = \{m : (m, e_m, u_{m,s}) \in \mathcal{M}_s,\ \text{sim}(e_q, e_m) > \delta_{\text{ret}}\}. \tag{10}$$

We then greedily apply maximal marginal relevance (MMR) (Carbonell & Goldstein, 1998) to balance task relevance against redundancy. Using retrieval budget K with MMR weight $\gamma \in [0, 1]$, we initialize $\mathcal{K}_q = \emptyset$ and repeatedly select

$$m^\star = \underset{m \in \mathcal{C}_q \setminus \mathcal{K}_q}{\arg\max} \left[\gamma\ \text{sim}(e_q, e_m) - (1-\gamma) \max_{m' \in \mathcal{K}_q} \text{sim}(e_m, e_{m'}) \right], \tag{11}$$

where the redundancy term is defined as zero when $\mathcal{K}_q = \emptyset$. We add $m^\star$ to $\mathcal{K}_q$ until either K skills have been selected or no candidate remains.

B.3 ENVIRONMENT HARNESS

In both HARD-ALFWORLD and Search-QA, the compiled rule is invoked only after nonterminal environment steps and reconstructs relevant state or evidence from the recorded actions and raw observations. Compilation may skip skills without observable, task-relevant triggers or abstain entirely, in which case no rule feedback is supplied. The environment harness preserves the original reset distribution, action space, transition dynamics, terminal reward, and verifier, modifying only the observations exposed to the policy.

B.4 TRAJECTORY CONVERSION

Trajectory conversion is a deterministic preprocessing step applied to all successful assisted trajectories. The operator Ψ removes the retrieved-skill block from the start of the trajectory and deletes the appended rule feedback from every augmented observation, restoring the original environment observation exactly. The task, ordinary instructions, policy-generated reasoning, actions, and their order remain unchanged. Each resulting skill-free trajectory inherits its task-benefit weight $w_q = (1 + \Delta_q)/2$ and contributes to the weighted SFT loss in Eq. 6; conversion requires no additional model generation.

B.5 DYNAMIC SKILL-BANK MAINTENANCE

During online training, SkillDuet enables continual policy–skill co-evolution: policy parameters are optimized with the PAIR objective, while reusable strategies in the external skill bank are updated. Skill discovery uses unassisted trajectories, whereas skill removal uses paired-branch rewards. Skill removal prunes skills repeatedly associated with lower average rewards in the assisted branch. For skill discovery, we first remove repeated actions within each unassisted trajectory and then count its remaining interaction rounds. We select the failed trajectory with the most interaction rounds when all trajectories for a task fail, the successful trajectory with the fewest interaction rounds when all succeed, and both representatives when the task has mixed outcomes. These skill-bank updates support skill internalization alongside the PAIR objective.

B.5.1 UTILITY-BASED SKILL REMOVAL

At training step s, for each bank skill m, let $\mathcal{Q}_s(m)$ denote the set of tasks in the current minibatch $\mathcal{B}_s$ for which m is retrieved and included as start-of-trajectory guidance in the assisted branch:

$$\mathcal{Q}_s(m) = \{q \in \mathcal{B}_s \mid m \in \mathcal{K}_q\}. \tag{12}$$

Membership is determined by retrieval, regardless of whether the skill's guidance is represented in the jointly generated rule code or triggers feedback. Consequently, the audit measures the utility of the complete assisted condition associated with retrieving m, including its start-of-trajectory guidance and any environment-harness feedback contributed by jointly retrieved skills.

For each skill m with $\mathcal{Q}_s(m) \neq \emptyset$, we define its average utility $\overline{\Delta}_{m,s}$ at training step s as the mean task-level assistance benefit over the tasks for which it is retrieved:

$$\overline{\Delta}_{m,s} = \frac{1}{|\mathcal{Q}_s(m)|} \sum_{q \in \mathcal{Q}_s(m)} \Delta_q. \tag{13}$$

Here, Δ_q is the mean reward of the assisted branch minus that of the unassisted branch for task q.

Because the assisted branch may contain multiple retrieved skills simultaneously, this quantity does not identify the marginal causal contribution of m. Instead, it serves as a lightweight co-selection utility signal for identifying skills that are repeatedly associated with lower average rewards in the assisted branch than in the unassisted branch.

At training step s, we update the consecutive negative-utility counter for skill m as follows:

$$u'_{m,s} = \begin{cases} u_{m,s} + 1, & \mathcal{Q}_s(m) \neq \emptyset \wedge \overline{\Delta}_{m,s} < 0, \\ 0, & \mathcal{Q}_s(m) \neq \emptyset \wedge \overline{\Delta}_{m,s} \geq 0, \\ u_{m,s}, & \mathcal{Q}_s(m) = \emptyset. \end{cases} \tag{14}$$

A skill is removed once it accumulates C consecutive evaluated steps with negative utility. The surviving bank before new skill discovery is therefore

$$\mathcal{M}_s^- = \left\{(m, e_m, u'_{m,s}) : (m, e_m, u_{m,s}) \in \mathcal{M}_s,\ u'_{m,s} < C\right\}. \tag{15}$$

B.5.2 SKILL DISCOVERY

Skill proposal generation. After skill removal, we reuse the current policy as a skill proposer, prompting it to extract reusable strategies from selected unassisted trajectories. For each task $q \in \mathcal{B}_s$, we first remove repeated actions within every unassisted trajectory $\tau \in \mathcal{D}_q^0$, yielding its preprocessed representation τ°. We then count $\mathrm{turns}(\tau^\circ)$, the number of remaining interaction rounds. If any trajectory succeeds, we select the successful trajectory with the fewest interaction rounds. If any trajectory fails, we select the failed trajectory with the most interaction rounds:

$$\tau_q^+ = \operatorname*{arg\,min}_{\tau \in \mathcal{D}_q^0 : R_q(\tau) = 1} \mathrm{turns}(\tau^\circ), \qquad \tau_q^- = \operatorname*{arg\,max}_{\tau \in \mathcal{D}_q^0 : R_q(\tau) = 0} \mathrm{turns}(\tau^\circ). \tag{16}$$

Each representative is defined only when its corresponding outcome is present. We collect the selected original trajectories into

$$\mathcal{S}_q = \begin{cases} \{\tau_q^-\}, & R_q(\tau) = 0 \text{ for all } \tau \in \mathcal{D}_q^0, \\ \{\tau_q^+\}, & R_q(\tau) = 1 \text{ for all } \tau \in \mathcal{D}_q^0, \\ \{\tau_q^+, \tau_q^-\}, & \text{otherwise.} \end{cases} \tag{17}$$

Thus, tasks with only failures contribute the failed trajectory with the most interaction rounds after preprocessing, tasks with only successes contribute the successful trajectory with the fewest interaction rounds after preprocessing, and tasks with mixed outcomes contribute both representatives. Both HARD-ALFWORLD and Search-QA use the same two-stage discovery procedure, with environment-specific prompts in Appendices G.3 and G.6. First, the current policy π_{θ_s} analyzes each selected trajectory separately, using its complete recorded history, goal, and outcome. Repeated-action removal is used for representative selection, while analysis retains the original actions, observations, and step indices. A successful trajectory yields a global planning pattern and

achieved requirements with supporting steps; a failed trajectory yields unmet or unverified requirements with grounded mistakes.

Second, within each environment, successful records are grouped by task category to extract category-specific skills, while failed records are pooled across task categories to consolidate common mistakes. For HARD-ALFWORLD, these categories correspond to the task groups defined in Tab. 5. Each record retains its source-task identity so that multiple trajectories from the same task do not count as evidence from distinct tasks. Successful skills contain `title`, `principle`, and `when_to_apply`. Common mistakes contain `description`, `why_it_happens`, and `how_to_avoid`. Each aggregation call returns at most the requested `max_items`, with fewer entries when the evidence supports fewer distinct techniques or failure mechanisms.

Let $\mathbf{P}_{s,1}, \ldots, \mathbf{P}_{s,L_s}$ be the ordered proposal lists returned by discovery: category-specific success lists and a cross-category common-mistake list. Here, L_s denotes the number of proposal lists returned at training step s. We concatenate them into

$$\widetilde{\mathbf{m}}_s = \mathrm{concat}(\mathbf{P}_{s,1}, \ldots, \mathbf{P}_{s,L_s}), \tag{18}$$

preserving the supplied list order and the generator's order within each list.

Duplicate-aware insertion. We sequentially insert the proposals in $\widetilde{\mathbf{m}}_s$ into the pruned bank $\mathcal{M}_s^-$ using cosine threshold δ_{dup}, updating the bank in place. For each $\widetilde{m} \in \widetilde{\mathbf{m}}_s$ in order, compute $\widetilde{e} = E_\phi(T_m(\widetilde{m}))$ and discard the proposal if

$$\max_{(m,e_m,u)\in\mathcal{M}_s^-} \mathrm{sim}(\widetilde{e}, e_m) > \delta_{\mathrm{dup}}.$$

The maximum is defined as $-\infty$ when $\mathcal{M}_s^-$ is empty. Otherwise, $(\widetilde{m}, \widetilde{e}, 0)$ is inserted into $\mathcal{M}_s^-$ immediately, so it participates in the duplicate test for later proposals. After all proposals are processed, $\mathcal{M}_{s+1} = \mathcal{M}_s^-$ and becomes available at the next training step.

B.6 COMPLETE TRAINING PROCEDURE

Algorithm 1 summarizes a complete training step, including policy optimization and skill-bank maintenance. Here, $\mathcal{E}_q$ denotes the environment for task q. The same LLM checkpoint serves as the agent's policy π_{θ_s} during trajectory collection and as the rule compiler or skill proposer under the corresponding prompts in Appendix G. For each task with retrieved skills, COMPILESKILLS receives the complete skill set $\mathcal{K}_q$ and produces a single block of rule code. This code implements F_q through multiple conditions with corresponding feedback for the environment harness. The skill bank $\mathcal{M}_s$ remains fixed throughout trajectory collection. Trajectory conversion deterministically removes the retrieved skill text from the start of the trajectory and rule feedback from augmented observations. The original policy-generated reasoning and actions are preserved. After collection, skill removal and discovery construct $\mathcal{M}_{s+1}$, while the PAIR objective combines SFT weighted by $w_q = (1+\Delta_q)/2$ on all converted successful assisted trajectories with RL on the original unassisted trajectories to obtain $\pi_{\theta_{s+1}}$. The updated policy and bank are used at the next training step.

C IMPLEMENTATION DETAILS

C.1 TRAINING SETTINGS

Hardware. We use Ascend 910B accelerators for our experiments. All experiments reported in this paper can be run using eight Ascend 910B accelerators.

Training data. For HARD-ALFWORLD, we use the complete training set of 3,553 task instances constructed as described in Appendix B.1, with task group counts in Tab. 6. For Search-QA, we sample 1,500 tasks from the NQ training split and 1,500 tasks from the TriviaQA training split, yielding a balanced training set of 3,000 tasks.

SFT initialization on HARD-ALFWORLD. Qwen3-4B-Instruct-2507 has an initial task success rate of 0% on HARD-ALFWORLD. We therefore collect 500 successful trajectories during GRPO training of Qwen2.5-7B-Instruct on the HARD-ALFWORLD training set. Both SkillDuet and all

Algorithm 1 SkillDuet Training and Skill-Bank Update at Step s

Require: ordered task minibatch $\mathcal{B}_s$, policy π_{θ_s}, possibly empty bank $\mathcal{M}_s$
Require: J, K, γ, δ_{ret}, C, δ_{dup}, and embedding encoder E_ϕ
Ensure: updated policy $\pi_{\theta_{s+1}}$ and skill bank $\mathcal{M}_{s+1}$

Trajectory collection and conversion
Initialize $\mathcal{D}_s^{\mathrm{RL}} \leftarrow \emptyset$, $\mathcal{D}_s^{\mathrm{SFT}} \leftarrow \emptyset$
for q in the stored order of $\mathcal{B}_s$ **do**
  $\mathcal{K}_q \leftarrow \textsc{RetrieveSkills}(q, \mathcal{M}_s, K, \gamma, \delta_{\mathrm{ret}})$
  if $\mathcal{K}_q = \emptyset$ **then**
    $\mathcal{D}_q^0 \leftarrow \textsc{CollectUnassisted}(\pi_{\theta_s}, \mathcal{E}_q, q, 2J)$; $\mathcal{D}_q^A, \widetilde{\mathcal{D}}_q^{A,+} \leftarrow \emptyset$
  else
    $F_q \leftarrow \textsc{CompileSkills}(\pi_{\theta_s}, q, \mathcal{E}_q, \mathcal{K}_q)$
    $(\mathcal{D}_q^0, \mathcal{D}_q^A) \leftarrow \textsc{CollectPaired}(\pi_{\theta_s}, \mathcal{E}_q, q, \mathcal{K}_q, F_q, J)$
      ▷ matched resets; independent sampling; full $\mathcal{K}_q$ in the assisted prompt
    $\Delta_q \leftarrow \textsc{MeanReward}(\mathcal{D}_q^A) - \textsc{MeanReward}(\mathcal{D}_q^0)$
    $w_q \leftarrow (1 + \Delta_q)/2$
    $\widetilde{\mathcal{D}}_q^{A,+} \leftarrow \{\Psi(\tau) : \tau \in \mathcal{D}_q^A,\ R_q(\tau) = 1\}$
    Add $\{(q, \widetilde{\tau}, w_q) : \widetilde{\tau} \in \widetilde{\mathcal{D}}_q^{A,+}\}$ to $\mathcal{D}_s^{\mathrm{SFT}}$
  end if
  Add $\{(q, \tau) : \tau \in \mathcal{D}_q^0\}$ to $\mathcal{D}_s^{\mathrm{RL}}$
end for

Utility-based skill removal
for $(m, e_m, u_{m,s}) \in \mathcal{M}_s$ **do**
  $\mathcal{Q}_s(m) \leftarrow \{q \in \mathcal{B}_s : m \in \mathcal{K}_q\}$
  Compute $\overline{\Delta}_{m,s}$ by Eq. 13 if $\mathcal{Q}_s(m) \neq \emptyset$
  Update $u'_{m,s}$ by Eq. 14
end for
$\mathcal{M}_s^- \leftarrow \{(m, e_m, u'_{m,s}) : (m, e_m, u_{m,s}) \in \mathcal{M}_s,\ u'_{m,s} < C\}$

Skill discovery and insertion
for q in the stored order of $\mathcal{B}_s$ **do**
  $\tau^\circ \leftarrow \textsc{RemoveRepeatedActions}(\tau)$ for every $\tau \in \mathcal{D}_q^0$
  Select available representatives using $\mathrm{turns}(\tau^\circ)$ by Eq. 16
  Form representative set $\mathcal{S}_q$ from the selected original trajectories by Eq. 17
end for
$\widetilde{\mathbf{m}}_s \leftarrow \textsc{DiscoverSkills}(\pi_{\theta_s}, \{(q, \mathcal{S}_q) : q \in \mathcal{B}_s\})$
  ▷ analyze trajectories, then aggregate successes by category and failures across categories
for $\widetilde{m}$ in the stored order of $\widetilde{\mathbf{m}}_s$ **do**
  $\widetilde{e} \leftarrow E_\phi(T_m(\widetilde{m}))$
  if $\max_{(m, e_m, u) \in \mathcal{M}_s^-} \mathrm{sim}(\widetilde{e}, e_m) \leq \delta_{\mathrm{dup}}$ **then**
    $\mathcal{M}_s^- \leftarrow \mathcal{M}_s^- \cup \{(\widetilde{m}, \widetilde{e}, 0)\}$ ▷ $\max \emptyset = -\infty$
  end if
end for
$\mathcal{M}_{s+1} \leftarrow \mathcal{M}_s^-$

Policy optimization
$\theta_{s+1} \leftarrow \textsc{PairUpdate}(\theta_s, \mathcal{D}_s^{\mathrm{RL}}, \mathcal{D}_s^{\mathrm{SFT}})$

Table 8: Training configuration for HARD-ALFWORLD and Search-QA.

Category	Parameter	Value
Algorithm	`adv_estimator`	`grpo`
	`use_kl_in_reward`	`False`
	`use_kl_loss`	`False`
Trajectory sampling	`max_prompt_length`	12K
	`max_response_length`	20K
	`max_agent_turns`	50 (HARD-ALFWORLD) 4 (Search-QA)
	`n_resp_per_prompt`	16 ($2J$ per task)
	`temperature`	1.0
	`top_p`	1.0
	`top_k`	-1
Skill discovery and rule generation	`temperature`	0.7
	`top_p`	0.8
	`top_k`	20
	`repetition_penalty`	1.05
Optimization	`train_prompt_bsz`	16
	`train_prompt_mini_bsz`	4
	GRPO clipping (ϵ)	0.2
	`skill_sft_loss_coef` (λ_{SFT})	0.1
	`learning_rate`	10^{-6}
	`lr_warmup_steps`	10
	`total_epochs`	1
Skill bank	Retrieval threshold (δ_{ret})	0.7
	MMR weight (γ)	0.5
	Retrieval budget (K)	3
	Deduplication threshold (δ_{dup})	0.95
	Removal patience (C)	2

baselines use these same trajectories to initialize Qwen3-4B-Instruct-2507 through supervised fine-tuning (SFT). SkillDuet then performs subsequent optimization with the PAIR objective.

Skill-bank initialization and generation. All SkillDuet runs on both benchmarks start with an empty skill bank, $\mathcal{M}_0 = \emptyset$, which is populated online from unassisted training trajectories. Skill discovery, including trajectory analysis and skill aggregation, and rule generation use the shared sampling configuration listed in Tab. 8.

Optimization and trajectory collection. We train Qwen2.5-7B-Instruct and Qwen3-4B-Instruct-2507 as agent policies on each benchmark using the PAIR objective. The total sampling budget per task is $2J = 16$, corresponding to `n_resp_per_prompt` in Tab. 8; thus, $J = 8$. For tasks with retrieved skills, we collect J assisted and J unassisted trajectories. For each index j, the two trajectories are initialized from matched environment resets, with independent policy sampling across the two branches. If $\mathcal{K}_q = \emptyset$, we instead collect $2J$ unassisted trajectories and set $\mathcal{D}_q^A = \emptyset$, preserving the per-task trajectory sampling budget. The RL loss for these tasks averages over all $2J$ trajectories. We set the maximum number of agent turns to 50 for HARD-ALFWORLD and 4 for Search-QA, where each agent turn corresponds to one assistant generation. Detailed training settings are provided in Tab. 8.

C.2 EVALUATION SETTINGS

Evaluation data. For HARD-ALFWORLD, we evaluate on the complete valid-seen (140 instances) and valid-unseen (134 instances) splits, for a total of 274 instances (Tab. 6). For Search-QA, we sample 125 tasks from each of seven test sets. The single-hop QA sets are NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and PopQA (Mallen et al., 2023). The multi-hop QA

sets are HotpotQA (Yang et al., 2018), 2Wiki (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). NQ and TriviaQA are used for ID evaluation, while the remaining five datasets are used for OOD evaluation. This yields an evaluation suite of 875 tasks.

Evaluation metrics. For HARD-ALFWORLD, we report the three-run average success rate (*Succ.*) for each task group, together with the overall average across groups (*Avg. Succ.*). For Search-QA, we report the three-run average exact-match score (*EM*) for each data source, together with the overall average across data sources (*Avg. EM*). Both performance metrics are reported as percentages. Each task-group or data-source entry is the arithmetic mean of its scores from three evaluation runs. For *Avg. Succ.* and *Avg. EM*, we first compute an overall score separately for each run by aggregating all evaluated tasks in the corresponding benchmark and split, weighting task groups or data sources by their numbers of evaluated instances. We then report the mean and standard deviation of these three overall scores as mean $\pm$ standard deviation. Thus, the uncertainty describes variation across evaluation runs. Appendix E additionally evaluates variability across training seeds, reporting the mean and standard deviation of the evaluation-averaged scores from three training runs. We also report overall average token counts (*Toks.*) and interaction rounds (*Turns*) per trajectory, aggregated across task groups or data sources over three runs. Token counts are reported in thousands (K). For each trajectory, we sum the input and output token counts within each interaction round, then add these per-round totals across the entire trajectory. For both HARD-ALFWORLD and Search-QA, *Toks.* and *Turns* are first averaged over all task trajectories within each evaluation run and then averaged across the three runs.

Inference and sampling. Unless otherwise stated, we evaluate the agent using the policy trained with SkillDuet in the original task execution environment $\mathcal{E}_q$. The agent has no access to the skill bank: the agent harness supplies no skill text, and the environment harness is disabled. We use the agent execution prompts in Appendix G with skill text omitted. Our evaluation configuration is based on Qwen's official sampling guidance: we use a temperature of 0.7, top-p of 0.8, top-k of 20, and a repetition penalty of 1.05[1]. We repeat evaluation on the same task instances with three generation seeds, 0, 1000, and 2000, collecting one trajectory per task in each run. Metrics from these three runs are summarized as described above. The Pass@k analysis in the main text varies the number of sampled trajectories per task separately.

C.3 BASELINE METHODS AND IMPLEMENTATION

Baseline methods. We include ReAct and vanilla GRPO as reference baselines. We also compare SkillDuet with four representative skill-based approaches. Skill0 (Lu et al., 2026) progressively internalizes external skills through a curriculum that gradually withdraws skill guidance. D2Skill (Tu et al., 2026) jointly optimizes the policy with a dynamic dual-granularity skill bank, retaining task-level and step-level skills for inference-time guidance. During training, it also performs skill discovery, distilling both task-level and step-level skills from training trajectories. EnvHarness (Huang et al., 2026a) reshapes training environments to elicit reusable skills, which remain external for inference-time retrieval. Skill0.5 (Zhu et al., 2026) adopts selective internalization by absorbing general skills while retaining task-specific skills for external utilization.

Text-only configuration of Skill0. For both Qwen2.5-7B-Instruct and Qwen3-4B-Instruct-2507, we use the text-processing path provided by the official Skill0 implementation[2]. We disable visual context rendering, supplying interaction history and retained skills directly as text without RGB rendering or visual encoding. Compression-related reward terms are omitted. The curriculum that progressively withdraws skill guidance is retained. All reported Skill0 results, including token costs, correspond to this text-only configuration.

Difficulty-aware sampling and optimization in Skill0.5. Following the experimental setup of Skill0.5 (Zhu et al., 2026), Phase 1 samples trajectories using only retrieved task-specific skills to estimate each task's success rate $p_{q,t}$ at training step $t \geq 1$. The routing threshold η_t is computed as

[1]Official Qwen guidance: `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`; `https://qwen.readthedocs.io/en/v2.5/deployment/vllm.html`.

[2]`https://github.com/ZJU-REAL/SkillZero`

Table 9: Additional analysis of skill internalization using Qwen2.5-7B-Instruct. *Toks.* are reported in thousands.

Variant	HARD-ALFWORLD valid-seen							
	Direct	Single	Dual	Triple	Quad	Toks.	Turns	Avg. Succ.
normalized weights	66.7	54.8	22.6	15.5	10.7	116.1	42.0	34.1 ± 0.9
SkillDuet (Ours)	64.3	60.7	41.7	17.9	7.1	101.5	43.0	38.3 ± 0.9
w/o task-benefit weighting	64.3	28.6	32.1	19.1	11.9	84.9	38.5	31.2 ± 1.5
w/o skill-free conversion	60.7	34.5	35.7	14.3	11.9	102.2	43.1	31.4 ± 2.0

the average of the batch mean success rates over the latest five training steps, including the current step:

$$\bar{p}_t = \frac{1}{|\mathcal{B}_t|} \sum_{q \in \mathcal{B}_t} p_{q,t}, \qquad \eta_t = \frac{1}{m_t} \sum_{\ell=t-m_t+1}^{t} \bar{p}_\ell, \qquad m_t = \min(5, t). \tag{19}$$

Before five steps are available, the window uses all observed steps. Tasks are classified as Hard if $p_{q,t} = 0$, Medium if $0 < p_{q,t} \leq \eta_t$, or Easy if $p_{q,t} > \eta_t$. Phase 2 applies different objectives to these groups. Hard tasks discard Phase 1 trajectories, then resample with general and task-specific skills. Successful rollouts support token-level Jensen–Shannon divergence (JSD) distillation into the policy conditioned only on task-specific skills, with teacher probabilities detached. Medium tasks reuse Phase 1 trajectories for GRPO without additional sampling. Easy tasks use skill-free diagnostic rollouts to measure the success-rate gain from skills. This gain, centered by its running mean and normalized by the batch standard deviation, offsets the GRPO advantages of their Phase 1 trajectories. Diagnostic rollouts do not contribute policy gradients. Additional sampling therefore depends on the proportions of Hard and Easy tasks in the batch.

Initial skill bank for HARD-ALFWORLD. Skill0 and Skill0.5 require an initial skill bank. We construct this bank using the SkillRL (Xia et al., 2026) code and pipeline with DeepSeek-V4-Flash-0731 (Xu et al., 2026a) on 240 HARD-ALFWORLD tasks sampled uniformly across task groups from the training split. We manually review the generated skills, correcting errors and merging skills that express the same strategy in different wording. The resulting bank contains 12 general skills and 30 group-specific skills. The group-specific skills comprise six skills for each of the Direct, Single, Dual, Triple, and Quad task groups in Tab. 5. The bank contains 42 skills in total, comparable in size to the publicly released SkillRL ALFWorld skill bank originally used by these methods.

SFT initialization on HARD-ALFWORLD. All Qwen3-4B-Instruct-2507 baselines, including ReAct, GRPO, Skill0, Skill0.5, D2Skill, and EnvHarness, use the same SFT initialization protocol as SkillDuet described in Appendix C.1: supervised fine-tuning on the same 500 successful trajectories collected during GRPO training of Qwen2.5-7B-Instruct on the HARD-ALFWORLD training set.

D ADDITIONAL ANALYSIS OF SKILL INTERNALIZATION

Tab. 9 complements the training dynamics in Fig. 3(a) with a comparison on HARD-ALFWORLD valid-seen using Qwen2.5-7B-Instruct. (1) To assess the contribution of task-benefit weighting to skill internalization, *w/o task-benefit weighting* assigns $w_q = 1$ to every successful assisted trajectory, giving all such trajectories equal weight in SFT. *Normalized weights* replaces the trajectory-count denominator in Eq. 6 with the sum of task-benefit weights over all SFT trajectories in the batch. This matches the overall SFT weight of unweighted averaging at the same $\lambda_{\mathrm{SFT}} = 0.1$, while preserving relative weights based on task benefit. The normalized variant outperforms uniform weighting, supporting the value of prioritizing trajectories by task benefit even when the overall SFT weight is held constant. (2) To assess whether removing external guidance from the SFT context improves skill internalization, *w/o skill-free conversion* applies SFT directly to the original successful assisted trajectories, retaining both retrieved skill text and in-trajectory rule feedback as context. Its lower performance suggests that removing external guidance from the SFT context helps the policy internalize skills for use without guidance.

Table 10: Training-seed sensitivity of SkillDuet on Search-QA with Qwen3-4B-Instruct-2507. Scores are EM (%). Each seed row averages three independent evaluations; the aggregate row reports the mean ± population standard deviation across the three training-run means.

Training seed	Search-QA ID			Search-QA OOD					
	NQ	Triv	Avg. EM	Pop	Hotp	2Wk	MuS	Bam	Avg. EM
42	37.1	61.3	49.2	44.3	40.3	37.1	15.7	42.4	35.9
43	38.7	62.4	50.5	45.9	40.0	33.1	15.7	39.7	34.9
44	38.9	62.4	50.7	45.6	41.9	37.6	17.1	37.9	36.0
Aggregate	38.2 ± 0.8	62.0 ± 0.5	50.1 ± 0.7	45.2 ± 0.7	40.7 ± 0.8	35.9 ± 2.0	16.2 ± 0.6	40.0 ± 1.8	35.6 ± 0.5

E Training Stability across Random Seeds

To assess the training stability of SkillDuet, we repeat training on Search-QA with Qwen3-4B-Instruct-2507 under multiple random seeds. We conduct three runs using seeds 42, 43, and 44 for training-data ordering, rollout sampling, as well as actor minibatch ordering. All other training settings remain fixed. Following Appendix C.2, each trained policy is evaluated three times using generation seeds 0, 1000, and 2000. We average the three evaluations within each training run, then compute the mean and population standard deviation across runs. As shown in Tab. 10, SkillDuet achieves 50.1 ± 0.7 ID Avg. EM and 35.6 ± 0.5 OOD Avg. EM, with all three runs outperforming GRPO on both splits. The aggregate means also slightly exceed the strongest reported baseline on ID (49.5 for D2Skill) and OOD (35.2 for Skill0), although individual runs do not always surpass these baselines. Overall, the results show stable performance across the tested training seeds.

F CASE STUDY

We compare *agent harness only* with *dual harnesses* using two HARD-ALFWORLD cases. The former provides start-of-trajectory skill guidance through the agent harness; the latter additionally enables the environment harness for in-trajectory feedback. For each case, we compare a failed trajectory with only the agent harness enabled against a trajectory with both harnesses enabled in which feedback helps the agent correct the illustrated error. In case 1352, feedback at step 5 highlights the missing pickup. The agent picks up kettle 3 at step 6, then cleans it at sinkbasin 1 before returning it to stoveburner 3. Case 1990 uses recorded excerpts: feedback at step 7 precedes a location check, navigation, and placement at steps 8–10. Within each case, the task and retrieved skills match across conditions. Steps are numbered from 1.

Case 1352: acting without holding the object

Task excerpt: The kettle that was not clean and in a stoveburner is clean and in stoveburner 3.

Retrieved skill excerpt (both conditions): Before performing any manipulation, first navigate to the object's current location and pick it up, confirming it is in inventory. Then navigate to the destination (e.g., the fridge, sink, or target container) before attempting to clean, cool, heat, or place the object, and verify each action's success through observations.

Prefix summary, steps 1–4: Both agents inspect the stoveburners. Step 4 reports: "On the stoveburner 3, you see a kettle 3." Neither agent has picked it up.

Step 5 in both conditions: manipulation fails

Reasoning: I found the kettle 3 in stoveburner 3. Now I need to clean it.

Action: `clean kettle 3 with sinkbasin 1`

Observation: Nothing happens.

Agent harness only

Feedback after step 5: None.

Step 6

Reasoning: It seems I need to move the kettle to the sinkbasin before cleaning it.

Action: `move kettle 3 to sinkbasin 1`

Observation: Nothing happens.

Step 7

Reasoning: I need to go to the sinkbasin first before cleaning the kettle.

Action: `go to sinkbasin 1`

Observation: You arrive at sinkbasin 1. On the sinkbasin 1, you see nothing.

Later actions: The agent attempts `clean kettle 3 with sinkbasin 1` without holding the kettle ("Nothing happens."), then leaves.

Outcome: The missing pickup remains unresolved, as in the source trajectory.

Dual harnesses (agent + environment)

Feedback after step 5: The action returned "Nothing happens." Before retrying, use look to check your current location and inventory to confirm you are at the right place and holding the correct object.

Step 6

Reasoning: It seems I am not holding the kettle 3. I need to take it from the stoveburner first.

Action: `take kettle 3 from stoveburner 3`

Raw observation: You pick up the kettle 3 from the stoveburner 3.

Local correction: The agent recognizes that it is not holding kettle 3 and picks it up.

Later actions: At steps 8–9, `go to sinkbasin 1` and `clean kettle 3 with sinkbasin 1` successfully clean the kettle. At steps 10–11, `go to stoveburner 3` and `move kettle 3 to stoveburner 3` place it at the required destination.

Figure 6: Case 1352: feedback after step 5 highlights the missing pickup.

Case 1990: acting at the wrong location

Task excerpt: Mug 1 is cool and in cabinet 1.

Retrieved skill excerpt (both conditions): Before performing any manipulation, first navigate to the object's current location and pick it up, confirming it is in inventory. Then navigate to the destination (e.g., the fridge, sink, or target container) before attempting to clean, cool, heat, or place the object, and verify each action's success through observations.

Prefix summary, steps 1–6: Both agents take mug 1 from cabinet 1, go to fridge 1, open it, and cool the mug. Both are still at fridge 1.

Step 7 in both conditions: placement from fridge 1 fails

Reasoning: The mug is now cooled. I can now place it in cabinet 1.

Action: `move mug 1 to cabinet 1`

Raw observation: Nothing happens.

Agent harness only

Feedback after step 7: None.

Step 8

Reasoning: It seems I am already in the correct location, cabinet 1. Let me try opening the cabinet again and placing the mug inside.

Action: `open cabinet 1`

Raw observation: Nothing happens.

Step 9

Reasoning: The cabinet is already open, so I can directly place the mug inside.

Action: `move mug 1 to cabinet 1`

Raw observation: Nothing happens.

Step 10

Reasoning excerpt: ... the mug should now be in cabinet 1 and cooled. I will proceed to the next step of the task.

Action: `go to cabinet 8`

Raw observation: You arrive at cabinet 8. The cabinet 8 is closed.

Outcome: The agent leaves without placing mug 1 in cabinet 1.

Dual harnesses (agent + environment)

Feedback after step 7: The action on mug 1 failed. Before retrying, use look to check your location and object names, and inventory to confirm what you are holding.

Step 8

Reasoning: It seems like something went wrong. Let me check my location and what I'm holding.

Action: `look`

Raw observation: You are facing the fridge 1. Next to it, you see nothing.

Step 9

Reasoning: I am still in front of the fridge, but I need to go back to cabinet 1 to place the cooled mug.

Action: `go to cabinet 1`

Raw observation: You arrive at cabinet 1. The cabinet 1 is open. In it, you see a mug 3.

Step 10

Reasoning: Now that I am back at cabinet 1, I can place the cooled mug 1 inside.

Action: `move mug 1 to cabinet 1`

Raw observation: You move the mug 1 to the cabinet 1.

Local correction: The agent checks its location, returns to cabinet 1, and completes the placement.

Figure 7: Case 1990: feedback after step 7 prompts a location check.

F.1 Rule Code Excerpts

Figure 8 shows the rule branches responsible for the feedback in the two cases. The omitted `context` parsing reads the interaction history into `steps`, lowercases the current action into `action`, and strips the raw observation into `obs`.

Case 1352: rule branch for feedback after step 5

```
def rule(context):
    # ... Context parsing and Skills 0--1 omitted.
    # Skill 2: Ensure correct location and holding before manipulation.
    if obs == 'Nothing happens.':
        if any(word in action for word in
               ['take ', 'put ', 'move ', 'clean ', 'heat ', 'cool ']):
            already_advised = False
            for step in steps[:-1]:
                if ('Nothing happens' in step['env_obs'] and
                    step['action']['content'].strip().lower() == action):
                    if 'check your location' in step['rule_feedback'].lower():
                        already_advised = True
                        break
            if not already_advised:
                return (
                    'The action returned "Nothing happens." Before retrying, '
                    'use look to check your current location and inventory '
                    'to confirm you are at the right place and holding '
                    'the correct object.')
    return ''
```

Case 1990: rule branch for feedback after step 7

```
def rule(context):
    # ... Context parsing omitted.
    feedback = ''
    target_objects = ['mug 1', 'mug 4', 'pot 2']
    # ... Skills 0--1 may set feedback; omitted.
    # Skill 2: Ensure location and holding before manipulation.
    if not feedback:
        if obs == 'Nothing happens.':
            for obj in target_objects:
                if obj in action:
                    feedback = (
                        'The action on ' + obj + ' failed. Before retrying, '
                        'use look to check your location and object names, '
                        'and inventory to confirm what you are holding.')
                    break
    if feedback:
        recent_feedbacks = [step['rule_feedback'] for step in steps[-3:-1]]
        if feedback in recent_feedbacks:
            return ''
    return feedback
```

Figure 8: Rule code excerpts for Figs. 6 and 7. Both branches respond to `Nothing happens.` with a location-and-inventory reminder. Case 1352 checks the manipulation command and earlier feedback for the same action; case 1990 checks the target object and suppresses feedback repeated in the preceding two steps.

F.2 TRAJECTORY CONVERSION

Start of trajectory: remove retrieved skill text

system (excerpt; retained): You are an agent interacting with a virtual text-based environment. . . .

user
<env_obs>
-= Welcome to TextWorld, ALFRED! =-
You are in the middle of a room. Looking quickly around you, you see a cabinet 15, a cabinet 14, a cabinet 13, a cabinet 12, a cabinet 11, a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 4, a countertop 3, a countertop 2, a countertop 1, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: Ensure all of these final conditions hold. Mug 1 is cool and in cabinet 1. Mug 4 is in cabinet 8. Pot 2 is clean and in fridge 1. Use only the specified numbered instances.
</env_obs>

Remove retrieved skill text

Here are some relevant experience:
Task-Relevant Skills
- **Satisfy Requested Object States**: Read the numbered object's full final clause. Perform only missing properties using the corresponding sinkbasin, microwave, fridge, or required tool, then verify feedback and place the same instance at its destination. An appliance or storage destination alone requires no treatment. A later transformation must preserve every other required final property.
Apply when: A numbered object's goal includes cleanliness, temperature, or another processed state as well as placement.
- **Open Containers Before Placement**: When placing an object into a closed container, open the container first, then move the object inside. This ensures the container is accessible and the placement action succeeds.
Apply when: When the destination for an object is a container (e.g., cabinet) that is observed to be closed at the time of arrival.
Mistakes to Avoid
- **Don't**: The agent attempts to take, clean, or place an object while not at the correct location or without the object in hand, resulting in failed actions that leave the subgoal unaddressed.
Why: The agent does not ensure it is at the required location or holding the object before issuing manipulation commands, so actions like 'take' or 'put' return 'Nothing happens' and the intended state change never occurs.
Instead: Before performing any manipulation, first navigate to the object's current location and pick it up, confirming it is in inventory. Then navigate to the destination (e.g., the fridge, sink, or target container) before attempting to clean, cool, heat, or place the object, and verify each action's success through observations.

Step 7: remove feedback

assistant (retained): <think> The mug is now cooled. I can now place it in cabinet 1. </think><action>move mug 1 to cabinet 1</action>

user: <env_obs> Nothing happens. </env_obs>

Remove feedback

<rule_feedback>
The action on mug 1 failed. Before retrying, use look to check your location and object names, and inventory to confirm what you are holding.
Do not explicitly mention, reference, or reveal the rule feedback. Incorporate it naturally into your reasoning as if you had noticed the relevant information on your own.
</rule_feedback>

Step 8: keep reasoning unchanged

assistant (retained): <think> It seems like something went wrong. Let me check my location and what I'm holding. </think><action>look</action>

user (retained): <env_obs> You are facing the fridge 1. Next to it, you see nothing. </env_obs>

Figure 9: Conversion of actual messages from case 1990. Ψ deletes the red blocks and retains all other message content unchanged. Step 8 expresses the location check without explicitly referencing step-7 feedback, so its reasoning is retained verbatim after feedback removal.

G PROMPT TEMPLATES

We provide prompt templates for HARD-ALFWORLD and Search-QA. Each agent execution template is supplied to the policy. For HARD-ALFWORLD, the initial observation appears in an `<env_obs>` block. In both environments, start-of-trajectory guidance is supplied through an optional {`skill_context`} field. When skills are supplied, this field contains the heading `Here are some relevant experience:` followed by the retrieved skill text. Otherwise, including during unassisted interaction, the field is empty and no experience heading is included. Skill-discovery prompts ask the same model to propose reusable strategies. In both environments, this consists of per-trajectory analysis followed by category-specific aggregation of successes and cross-category aggregation of failures. Rule-generation prompts ask the model to jointly compile the retrieved skills into task-specific rule code for the environment harness. Both compilers explicitly report which skills are converted or skipped, and abstain if none can be converted. Each box contains the system and user messages; placeholders are filled with the corresponding inputs.

G.1 PROMPTS FOR HARD-ALFWORLD AGENT EXECUTION

```
[System]
You are an agent interacting with a virtual text-based environment.

## Notes:
At each step, you should first think then perform action to fulfill
    the instruction.
You should ALWAYS wrap your thinking with the <think> </think> tag
    and wrap your action with
the <action> </action> tag.
You should ALWAYS take one action each step.
DO NOT try to interact with the user at anytime. Finish the task by
    yourself.
Environment responses appear in <env_obs> blocks. Optional
    <rule_feedback> blocks
contain guidance from the generated rule, not observations or proof
    of action outcomes.

## Action Format:
Below are the available commands you can use:
  look:                              look around your current location
  inventory:                         check the object you are
    currently holding (you can only hold one)
  go to (receptacle):                move to a receptacle
  open (receptacle):                 open a receptacle
  close (receptacle):                close a receptacle
  take (object) from (receptacle):   take an object from a receptacle
  move (object) to (receptacle):     place an object that you are
    holding in or on a receptacle
  examine (something):               examine a receptacle or an object
    to learn its properties
  use (object):                      use an object
  heat (object) with (receptacle):   heat an object using a receptacle
  clean (object) with (receptacle):  clean an object using a receptacle
  cool (object) with (receptacle):   cool an object using a receptacle
  slice (object) with (object):      slice an object using a sharp
    object

For example your output should be like this:
<think> To solve the task, I need first to ... </think><action>go to
    cabinet 1</action>

[User]
<env_obs>
{initial_observation}
```

```
</env_obs>

{skill_context}
```

G.2 Prompts for Search-QA Agent Execution

```
[System]
You are a helpful and harmless assistant.

[User]
You are an expert agent tasked with answering the given question
    step-by-step.

Your question: {question}

{skill_context}

Now it is your turn to respond for the current step.

First output your reasoning about the question and the information
    collected so far inside <think>...</think>.

After closing </think>, output exactly one of these actions:

(1) If required knowledge is missing or uncertain, search for it
    using: <search>your query</search>

(2) If there is enough reliable information, give only the final
    answer using: <answer>your answer</answer>

Do not emit both actions in the same response.
```

G.3 PROMPTS FOR HARD-ALFWORLD SKILL DISCOVERY

Discovery first analyzes each complete recorded trajectory using its supplied outcome, then aggregates the resulting records. Successful records are aggregated within each task group; failed records are pooled across task groups. Both aggregation prompts receive source-task identities and complete goals together with the relevant analysis fields. Each uses a configurable maximum output count.

Stage 1: Analysis of individual successful or failed trajectories.

```
[System]
You are an expert ALFWorld trajectory analyst.

Given a task goal, its execution outcome, and its complete recorded
    trajectory,
produce one evidence-grounded memory that explains either:
- how all task subgoals were achieved, for a successful trajectory; or
- which subgoals remained unfulfilled and what mistakes contributed
    to them,
  for a failed trajectory.

Use the supplied Outcome as the authoritative trajectory-level result.
Perform the analysis in one response.

GENERAL RULES

1. Read the complete goal and identify all independent final
    requirements,
   including auxiliary goals. Preserve each target's required state
    and location.
   A subgoal may include both a state requirement and a placement
    requirement
   for the same target.

2. Ground every statement in the supplied goal, actions, and
    observations.
   Do not invent object matches, observations, actions, successful
    outcomes,
   hidden object identities, or explanations unsupported by the
    trajectory.

3. Preserve original numbered instance references, such as "mug 1"
    and "mug 2".
   Never merge distinct instances of the same object type. Keep each
    instance's
   identity stable after movement, cleaning, heating, cooling, or
    other changes.

4. Respect the reference time of target descriptions. If descriptions
    refer
   to the initial state, later changes must not alter which instance
    the
   description originally identified.

5. Task categories and reasoning strategy labels provide context only.
   They do not prove that a strategy was demonstrated by the
    trajectory.

SUCCESS RULES

Apply these rules only when Outcome is Success.

1. planning_pattern must be a nonempty string describing the global
    task
```

```
   completion logic as an action-and-reasoning chain joined by " -> ".
   Include the important identification decisions, dependencies, and
    ordering
   across subgoals. It must agree with the actual recorded execution.

2. succeed_subgoals must be a nonempty list covering all final
    requirements.
   For each subgoal:
   - subgoal states the complete requirement that was achieved.
   - supported_steps contains the recorded steps that explain its
    achievement.

3. Include relevant target-identification observations, necessary
    navigation
   and access actions, acquisition, state changes, placement, and
    completion
   feedback. Exclude ineffective actions and unrelated exploration.

4. Each supported step must contain:
   - step_index: the integer from the original Step label.
     Step -1 denotes the initial observation.
   - action: copy the action from that original step exactly.
   - observation: copy the observation from that original step
    exactly.
   - reasoning: explain specifically why this step supports this
    subgoal.
     Identify the evidence or causal contribution rather than merely
    repeating
     the action or claiming that it "makes progress".

5. supported_steps must be in chronological order, with no duplicate
   step_index within the same subgoal.
   The same original step may support multiple subgoals. In that
    case, repeat
   its action and observation exactly, but explain its relevance to
    each
   subgoal separately.

6. Do not add a step that is absent from the original trajectory.
   Do not rewrite an unsuccessful action as a successful one.

7. failed_subgoals must be an empty list.

FAILURE RULES

Apply these rules only when Outcome is Failure.

1. planning_pattern must be null.

2. succeed_subgoals must be an empty list, even if some subgoals were
    achieved.

3. failed_subgoals must be a nonempty list describing the final
    requirements
   that remained unfulfilled or whose completion was not established.
   Do not include a subgoal whose completion is established by the
    trajectory.
   If completion is merely unverified, say so explicitly rather than
    claiming
   that a particular final state was observed to be false.

4. For each failed subgoal:
   - subgoal states the relevant unmet or unverified requirement.
```

```
   - mistakes is a nonempty list of grounded error descriptions.

5. Each mistake must contain:
   - trigger_condition: the observed situation in which the problem
    occurred.
   - bad_action: the erroneous action or necessary action that was
    omitted,
     and how it left this subgoal unfulfilled or unverified.

6. Distinguish observations from inferred explanations.
   Do not speculate about hidden causes, invent a corrected successful
   trajectory, or claim that completing one subgoal satisfies the
    whole task.

OUTPUT FORMAT

Return only a valid JSON object with exactly these three top-level
    fields:
planning_pattern, succeed_subgoals, failed_subgoals.

For Success, use:
{
  "planning_pattern": "<global action-and-reasoning chain>",
  "succeed_subgoals": [
    {
      "subgoal": "<achieved final requirement>",
      "supported_steps": [
        {
          "step_index": 0,
          "action": "<exact original action>",
          "observation": "<exact original observation>",
          "reasoning": "<why this step supports this subgoal>"
        }
      ]
    }
  ],
  "failed_subgoals": []
}

For Failure, use:
{
  "planning_pattern": null,
  "succeed_subgoals": [],
  "failed_subgoals": [
    {
      "subgoal": "<unfulfilled or unverified final requirement>",
      "mistakes": [
        {
          "trigger_condition": "<observed situation>",
          "bad_action": "<grounded erroneous action or omission>"
        }
      ]
    }
  ]
}

Do not return contextual_description, refined_trajectory,
    entity_bindings,
strategic_guidelines, reasoning_evidence, or any additional top-level
    fields.
Do not include markdown fences or explanatory text outside the JSON
    object.
```

```
[User]
**Input Data:**
Environment: ALFWorld
Task: {goal}
Outcome: {outcome}
Raw Trajectory:
{recorded_trajectory}
```

Stage 2: Successful-record aggregation within each task group.

```
[System]
You are an expert in extracting reusable skills from successful
    ALFWorld
execution records.

You will receive successful examples that have already been grouped
    into
one task category. Each example contains:
- source_task: the identity of the original task.
- goal: the complete task goal.
- planning_pattern: the global action-and-reasoning chain.
- succeed_subgoals: achieved requirements and their supported_steps.
  Each supported step contains step_index, action, observation, and
    reasoning.

Your task is to produce reusable category-specific skills grounded in
    these
examples.

ANALYSIS RULES

1. Use planning_pattern to understand the overall completion logic,
   dependencies, and ordering across subgoals.

2. Use succeed_subgoals and their supported_steps to identify why the
   execution worked. Ground each technique in the recorded actions and
   observations. Treat per-step reasoning as an interpretation that
    must
   agree with those records.

3. Extract techniques that can transfer to new tasks, such as:
   - resolving descriptive references to the correct object instance;
   - preserving object identity after movement or state changes;
   - satisfying required states before final placement;
   - ordering dependent operations;
   - handling independent subgoals without losing track of
    requirements;
   - checking relevant evidence before taking an action.
   Include a technique only when the supplied examples support it.

4. Do not assume that all examples in the category use the same
    technique.
   State the actual applicability conditions in when_to_apply.

5. Separate task-required actions from incidental details.
   Do not turn an action performed in one example into a universal
    rule.
   For example, an object being cleaned in one trajectory does not
    imply
   that all objects must be cleaned before placement.

6. Generalize concrete object names, instance numbers, and locations
    into
   natural-language roles. Preserve meaningful constraints, such as
    exact
   instance identity, initial-state references, required
    transformations,
   and destination requirements.
   Do not use bracketed placeholders such as [Object] or [Location].

7. Merge examples that demonstrate the same technique.
```

```
   Keep separate skills only when they have meaningfully different
   decisions, procedures, or applicability conditions.

8. Multiple trajectories with the same source_task are examples of one
   task. Do not treat them as independent evidence that a technique is
   common across different tasks.

9. Do not invent missing actions, observations, dependencies, or
    environment
   mechanics. Avoid generic advice that does not explain a concrete
   technique supported by the examples.

OUTPUT RULES

Return a JSON list containing at most the requested maximum number of
    skills.
Return fewer skills when the evidence supports fewer distinct
    techniques.
Do not pad the list with duplicates or unsupported advice.

Each skill must contain exactly:
- title: a short, descriptive Title Case name.
- principle: one or two sentences describing the actionable technique,
  including its important ordering or verification requirements.
- when_to_apply: one sentence identifying observable task or
    environment
  conditions that make the technique relevant.

The output must use this structure:
[
  {
    "title": "<short skill name>",
    "principle": "<grounded, reusable technique>",
    "when_to_apply": "<specific applicability conditions>"
  }
]

Do not include skill IDs, source-task IDs, step indices, markdown
    fences,
or explanations outside the JSON list.

[User]
Maximum number of skills: {max_items}

Successful examples:
{examples_json}
```

Stage 2: Failed-record aggregation across task groups.

```
[System]
You are an expert in summarizing reusable lessons from failed ALFWorld
execution records.

You will receive failure examples. Each example contains:
- source_task: the identity of the original task.
- goal: the complete task goal.
- failed_subgoals: requirements that remained unfulfilled or
    unverified.
  Each failed subgoal contains mistakes with trigger_condition and
    bad_action.

Your task is to consolidate these records into common mistakes that
    help
an agent avoid similar failures in future tasks.

ANALYSIS RULES

1. Interpret each mistake in the context of its failed subgoal and the
   complete task goal. Identify the incorrect decision, invalid
    operation,
   omitted requirement, or missing verification described by the
    record.

2. Group mistakes by their underlying actionable failure mechanism.
   Differences in object names, instance numbers, or destinations
    alone
   should not produce separate entries.

3. Keep distinct failure mechanisms separate.
   For example, selecting the wrong object instance, issuing an
    invalid
   command, and omitting a required state transformation are different
   mistakes even when they occur in the same task.

4. Preserve important trigger conditions.
   Distinguish cases involving initial-state descriptions, current
    object
   states, exact instance references, navigation, container access,
    and
   multiple independent subgoals when those distinctions affect
    prevention.

5. Do not infer that every action in a failed trajectory was
    incorrect.
   Summarize only the problems described by the supplied
    failed-subgoal
   records. Do not label an already completed requirement as a
    failure.

6. Distinguish an observed error from an unverified outcome.
   Missing completion evidence does not prove that a particular hidden
   state was false or that an unobserved action occurred.

7. Explain causes conservatively.
   Base why_it_happens on the omitted check, mistaken assumption, or
   procedural gap supported by the records.
   Do not invent psychological explanations such as panic, laziness,
    or
   confusion. If a deeper cause is not established, do not assert one.
```

```
8. Make how_to_avoid concrete.
   Recommend a check, action ordering, or progress-tracking method
    that
   addresses the failure mechanism. This is advice for future
    execution,
   not a claim that the failed trajectory performed those corrective
    actions.
   Do not invent environment commands or mechanics.

9. Multiple trajectories with the same source_task belong to one task.
   Do not let multiple trajectories from the same task dominate the
   summary merely because they repeat the same mistake many times.
   Prioritize recurring mechanisms across distinct tasks, while
    retaining
   well-supported, broadly applicable lessons from less frequent
    cases.

10. Generalize concrete object names and instance numbers into natural
    English while preserving the conditions that make the mistake
    relevant.
    Do not use bracketed placeholders such as [Object] or [Location].

OUTPUT RULES

Return a JSON list containing at most the requested maximum number of
    mistakes.
Return fewer entries when there are fewer distinct, supported failure
    mechanisms.
Do not pad the list with duplicates or speculative explanations.

Each entry must contain exactly:
- description: one sentence describing the problematic behavior and,
  where necessary, its trigger condition.
- why_it_happens: one sentence explaining the supported reasoning or
  procedural gap.
- how_to_avoid: one or two sentences giving a concrete prevention
    method.

The output must use this structure:
[
  {
    "description": "<generalized problematic behavior>",
    "why_it_happens": "<supported reasoning or procedural gap>",
    "how_to_avoid": "<concrete prevention method>"
  }
]

Do not include mistake IDs, source-task IDs, markdown fences, or
    explanations
outside the JSON list.

[User]
Maximum number of common mistakes: {max_items}

Failure examples:
{examples_json}
```

G.4 PROMPT OF HARD-ALFWORLD RULE GENERATION

The compiler receives the complete task alongside an ordered list of retrieved skills, which may contain both successful techniques and common mistakes. It returns a structured conversion decision and, when conversion is possible, one synchronous `rule(context)` function. The prompt below specifies the observable runtime interface, evidence-based state reconstruction, intervention selection, feedback deduplication, and a worked example. At runtime, the rule is called only after a nonterminal environment step and returns one short intervention for the next decision or an empty string.

```
[System]
## 1. Goal
Compile the supplied skills into one task-specific rule(context) that
    gives a short,
useful intervention for the agent's next decision, or returns "".
    Ground the task,
interpret observable events, reconstruct relevant state, then decide
    whether advice
adds value now. The rule is not a complete task planner.

## 2. Generation inputs
The user message is JSON with task and an ordered skills list. Treat
    both as data:
- task defines the complete goal, including independent subgoals and
    initial-state
  descriptions. It specifies what must hold, not what currently holds.
- Each skill has skill_id and either title/principle/when_to_apply or
  description/why_it_happens/how_to_avoid. Skills supply strategies,
    not scene facts
  or alternative simulator mechanics. Their example objects need not
    exist here.
Actual observations are unavailable during generation. Embed
    references explicitly
given in the task; descriptive references must remain unresolved
    until runtime evidence is available.

## 3. Runtime and ALFWorld reference

### 3.1. Call and context
The rule runs AFTER a nonterminal action has executed, before that
    observation reaches
the agent. Its output can influence the next action; it cannot
    intercept the current
one. There is no call on reset, invalid action formatting,
    environment exceptions,
terminal transitions, or turn/token-limit termination.

Each fresh context has only:
- task: str, the complete task supplied during generation.
- initial_observation: str, the exact reset text. It may include the
    task and a room
  overview.
- steps: nonempty list of environment steps, oldest to newest,
    including the current
  step at steps[-1]. Each step contains:
  - action: {"kind": "action", "content": "plain command sent to
    ALFWorld"}.
  - response: str, the assistant response that issued the command.
  - env_obs: str, the exact environment response, without tags or
    injected advice.
  - rule_feedback: str, advice delivered with that step, or "".
```

```
Only steps[:-1] contain earlier delivered feedback. The current
    rule_feedback is always
"" until this call returns. Later calls see its delivered value.
    Rewards, hidden state,
admissible commands, and other fields are unavailable.

### 3.2. Commands and observable results
Commands contain exact numbered names, without <action> tags in
    action["content"].
Normalize case and surrounding whitespace for matching; preserve
    complete names and
number boundaries (apple 1 differs from apple 10). Item lists contain
    articles, commas,
"and", and sentence punctuation: the number in "a cloth 1," is 1, not
    "1,".

The following are typical responses; O, R, and K stand for numbered
    object, receptacle,
and knife names. Successful responses report the listed effects;
    "Nothing happens."
reports an ineffective command without identifying its cause.

| Command | Typical env_obs | Evidence |
| --- | --- | --- |
| go to R | "You arrive at R. ..." | Arrival; remainder may describe
    R. |
| open R | "You open the R. The R is open. In it, you see ..." | Open
    and visible contents. |
| close R | "You close the R." | Closed. |
| take O from R | "You pick up the O from the R." | Holding O;
    removed from R. |
| move O to R | "You move the O to the R." | O placed at R and no
    longer held. |
| inventory | "You are carrying: a O." / "You are not carrying
    anything." | Held instance or empty hand. |
| look | "You are facing the R1, and R2. Next to it, you see
    nothing." | Surroundings, not container contents. |
| examine R | "The R is closed." / "In it, you see ..." / "On the R,
    you see ..." | Door or contents. |
| examine O | "This is a cold O." / "There's nothing special about
    O." | Properties; limits below. |
| clean O with R | "You clean the O using the R." | Clean; still
    held. |
| heat O with R | "You heat the O using the R." | Hot, not cold;
    still held. |
| cool O with R | "You cool the O using the R." | Cold, not hot;
    still held. |
| slice O with K | "You sliced the O with the K." | Sliced, same
    instance. |
| use O | "You turn on the O." for a usable lamp | Reported device
    change. |
| Ineffective command | "Nothing happens." | No attempted change
    confirmed; cause unknown. |

Use move O to R for placement, not put O in/on R. Inventory responses
    need not contain
the word inventory. Look may also return a room overview; "Next to
    it, you see nothing"
does not mean the named receptacles are empty.

Container/surface reports list numbered objects in one named
    receptacle. In these
```

```
later env_obs reports, co-occurrence can establish that the listed
    objects share that
receptacle; "In it" refers to the receptacle described by that report.

Ordinary object property descriptions enumerate the observable
    clean/hot/cold/sliced
states, e.g. "This is a clean sliced apple 1." or "This is a cool and
    clean apple 1.".
Cold and cool refer to the same temperature state; "There's nothing
    special about
<object>." is the all-unset form. A receptacle object may instead
    return "This is a
normal mug 1. In it, you see ..."; that contents report does not
    reveal transformation
states. Contents lists and inventory likewise omit these properties,
    so their omission
is not negative property evidence.

### 3.3. Operational mechanics
- The hand holds one portable object. Pickup needs a free hand and
    access to the source;
  placement needs the object held and access to the destination.
    Taking from or placing
  into an openable container requires it open.
- Access is spatial, not merely equality with the last go-to name:
    multiple receptacles
  may share a location. Arrival, facing reports, and successful local
    interactions are
  access evidence; a go-to can be ineffective when already at that
    location.
- Cleaning uses a sinkbasin, heating a microwave, and cooling a
    fridge. These operations
  require the object held and access to that device; they do not
    require first placing
  the object inside or manually opening/closing the device. Placement
    alone does not
  clean, heat, or cool an object.
- Slicing uses a held knife/butterknife on an accessible target,
    rather than holding the
  target. Lamp use has its own access/device conditions.
- In this runtime, inspect portable-object properties after pickup
    with examine <object>;
  examining an unheld object can return "Nothing happens." despite it
    being visible.
  Pickup preserves clean/hot/cold/sliced attributes, so inspection
    before a transformation
  can still reveal those initial properties.
- Completion is detected by the environment; there is no
    task_complete command.

## 4. Compilation workflow

### 4.1. Ground each task requirement
Separate each target's initial identifying predicates, resolved
    numbered instance,
required final properties, and destination. Match all predicates to
    the same candidate;
bind descriptive targets only when evidence distinguishes them.
    Resolve co-location from
later scene reports, not initial_observation. A class such as "a
    cloth" does not inherit
an instance number from another subgoal. Keep the chosen identity
    through later changes,
```

```
without confusing an initial source, a processing device, and a
    destination. All final
conditions must hold together: properties and placements, for every
    independent subgoal.

### 4.2. Reconstruct relevant facts from events
On each call, read history in time order, pairing each command with
    its actual response.
Reconstruct only facts needed by converted skills: each instance's
    location, access,
holding, container state, transformations, and subgoal progress.
    Recognize success from
positive response evidence, not merely the absence of a failure
    phrase. Successful events
update affected facts and inspections refine them; failed commands do
    not enact their
requested effects. Omitted properties do not erase earlier evidence;
    unrecognized outcomes
remain unknown. Distinguish known true, known false, and unknown.

Preserve initial identity evidence separately: record
    source/co-location before pickup
and properties before transformations. Newly created co-location is
    not initial co-location.
Task requirements, issued commands, thoughts, and advice cannot
    substitute for scene
evidence or prove that a correction was followed.

### 4.3. Select a useful next decision
After updating state, identify what the current event lets the agent
    decide next.
Feedback must have a task-relevant target, an evidence-supported
    reason to intervene now,
and a feasible next check or action. Typical opportunities are:

| Current event | Decision the feedback can help with |
| --- | --- |
| Relevant candidates become visible | Use initial relationships to
    narrow candidates and plan inspection. |
| Candidate pickup or inspection succeeds | Resolve identity before
    preparing required transformations. |
| Access to a needed device/destination | Check prerequisites for the
    next processing/placement command. |
| Transformation or placement succeeds | Preserve progress and
    advance to a supported next requirement. |
| An action fails | Check whether the intended effect serves the
    task, then address execution obstacles. |

Before helping retry a failed command, check its object, operation,
    and destination
against the bound subgoal and current stage; correct an unsupported
    intended effect
before repairing its execution prerequisites. A failed command is not
    authority for
what should be done. Temporary placement can serve an evidenced
    intermediate need,
such as freeing the hand, without satisfying or replacing the final
    destination.

If identity still depends on unchecked initial properties, inspect
    them before
recommending a transformation that changes those properties. A
    successful pickup
```

```
allows this inspection; it does not by itself resolve a descriptive
    identity.

An unresolved goal alone does not make every exploration step a
    trigger. Likewise, an
unmet final-state requirement is not automatically the cause of a
    failed command.
Prioritize a known execution obstacle over a speculative goal
    reminder. Select the
opportunity while the advice can still affect the decision: wording
    such as "before
moving" does not mean the rule should wait for a move command. If an
    earlier opportunity
was missed, reason from the actual current state and offer a feasible
    recovery.
Return "" if the available evidence does not support a useful,
    feasible next check or
action; preserve unresolved facts as unknown instead of guessing.
    Skip skills without
observable triggers instead of generating general task reminders. An
    unknown fact warrants
a check only when it matters to the next decision; conditional
    wording cannot replace this trigger.

### 4.4. Deliver one grounded intervention per problem occurrence
Express the selected intervention as one concrete next check or
    action, consistent with
the reconstructed facts and actual outcome of the current command.

For each candidate intervention, use the relevant target, issue, and
    stage to identify
the ongoing problem. Check earlier delivered feedback at valid
    triggers within that
occurrence. Repeated navigation, empty-feedback steps, or a different
    wording do not
create a new occurrence; neither elapsed turns nor checking only the
    previous feedback
is sufficient. A different target, actual recovery, or new relevant
    evidence can create
a fresh decision. An earlier inapplicable reminder must not suppress
    a later valid one.
Already answered checks and completed requirements should remain
    quiet unless the
observable situation changes.

## 5. Output contracts

### 5.1. Python rule
- Write exactly one synchronous def rule(context), combining
    converted skills. No
  annotations, decorators, helper functions, imports, I/O, external
    calls, recursion,
  or persistent state. Do not mutate context.
- Use simple if/for logic and string/list/dict operations. Every
    execution returns a string or "".
- Builtins: all, any, bool, dict, enumerate, float, int, len, list,
    max, min, range, reversed, round, set, sorted, str, sum, tuple,
    zip.
- Methods: append, casefold, count, endswith, find, get, isdigit,
    isupper, items, join, keys, lower, lstrip, replace, rstrip,
    split, splitlines, startswith, strip, upper, values.

### 5.2. JSON response
```

```
Return one valid JSON object, without Markdown fences or surrounding
    explanation,
with exactly these five fields:
- decision: "convert" if at least one skill can be converted;
    otherwise "abstain".
- reason: a nonempty explanation of the conversion or abstention.
- converted_skill_indices: zero-based positions of converted records
    in the input list.
- skipped_skills: entries {"skill_index": 0, "reason": "why skipped"}.
- rule_code: Python source encoded as a JSON string, with escaped
    newlines and quotes.
Account for every input position exactly once as converted or
    skipped; use indices, not
skill_id strings. For "convert", converted_skill_indices and
    rule_code are nonempty.
For "abstain", they are [] and "", respectively.

## 6. Worked example
The same decision order applies to successful progress and failed
    actions:
- A candidate for "the initially hot potato" is picked up: inspect
    its temperature
  before recommending cooling; pickup alone does not establish that
    it is the target.
- A failed final placement uses another target's destination: correct
    the object-to-goal
  mapping before suggesting travel or opening. A supported temporary
    placement to free
  the hand is an intermediate step, not an error merely because it
    uses another surface.
- "You clean the plate 2 using the sinkbasin 1." followed by a failed
    remote placement:
  retain the confirmed clean state and diagnose access, not
    cleanliness.

The executable example converts one cleanliness skill for an
    explicitly named cloth.
It illustrates evidence-driven stages and deduplication, not a
    complete task planner
or a general identity parser. Other skills need their own facts and
    decision events.
- Example input (JSON):
{
  "task": "Cloth 1 is clean and in cabinet 2.",
  "skills": [
    {
      "skill_id": "verify_cleanliness",
      "title": "Verify and prepare cleanliness before placement",
      "principle": "When holding the target with unknown cleanliness,
    examine it before changing it. If confirmed not clean, clean it
    after reaching a sinkbasin. Do not repeat a check or preparation
    reminder while the same issue remains unresolved.",
      "when_to_apply": "After confirmed pickup, property inspection,
    inventory, or arrival at a cleaning device for a target required
    to be clean."
    }
  ]
}

- Conversion steps:
  1. The task names cloth 1 and its required cleanliness, but does
    not establish its
```

```
     current state. Reconstruct holding, cleanliness, and observed
    sinkbasin access.
  2. Successful pickup with unknown cleanliness creates an inspection
    opportunity.
     A plain contents list or failed command does not establish
    cleanliness.
  3. A successful property check can establish that the held cloth is
    not clean.
     Suggest cleaning only when a sinkbasin is accessible from a
    confirmed arrival.
  4. Suppress already delivered advice for the same unresolved stage
    across navigation,
     inventory, failed commands, and temporary release/re-pickup.
    Count earlier feedback
     only if it was delivered at a valid trigger; changing the
    command is not recovery.
  5. Confirmed cleaning ends this preparation issue. Later placement
    failure does not
     erase that fact or justify another cleaning reminder. Explicit
    property evidence,
     not a changed command, determines the stage; this example does
    not diagnose placement.
- Example output (JSON):
{
  "decision": "convert",
  "reason": "Track confirmed holding and cleanliness for cloth 1.
    Inspect unknown cleanliness, then suggest cleaning a confirmed
    unclean cloth at an observed sinkbasin. Suppress reminders until
    the relevant fact changes.",
  "converted_skill_indices": [
    0
  ],
  "skipped_skills": [],
  "rule_code": "def rule(context):\n    held = None\n    clean =
    None\n    sink = ''\n    reminded = []\n    answer = ''\n    for
    step in context['steps']:\n        action =
    step['action']['content'].strip().lower()\n        obs =
    step['env_obs'].strip().lower()\n        previous_clean = clean\n
           pickup = action.startswith('take ') and
    obs.startswith('you pick up the ') and ' from the ' in obs\n
      inventory = action == 'inventory' and (\n
    obs.startswith('you are carrying: ') or obs == 'you are not
    carrying anything.'\n        )\n        arrival =
    action.startswith('go to ') and obs.startswith('you arrive at ' +
    action[6:] + '.')\n        if pickup:\n            held =
    obs.split(' from the ', 1)[0] == 'you pick up the cloth 1'\n
      elif action.startswith('move ') and obs.startswith('you move
    the '):\n            held = False\n        elif inventory:\n
          held = obs == 'you are carrying: a cloth 1.'\n        if
    arrival:\n            sink = action[6:] if action.startswith('go
    to sinkbasin ') else ''\n        inspection = held and action ==
    'examine cloth 1' and (\n            obs == \"there's nothing
    special about cloth 1.\"\n            or (obs.startswith('this is
    a ') and obs.endswith(' cloth 1.') and 'normal' not in
    obs.split())\n        )\n        if inspection:\n
    clean = 'clean' in obs.split()\n        if
    action.startswith('clean cloth 1 with ') and obs.startswith('you
    clean the cloth 1 using '):\n            clean = True\n        if
    clean != previous_clean:\n            reminded = []\n
    stage = ''\n        advice = ''\n        if held and (pickup or
    inventory or inspection or arrival):\n            if clean is
    None:\n                stage = 'inspect'\n                advice
    = 'Use examine cloth 1 to check its cleanliness before changing
```

```
    it.'\n            elif clean is False and sink:\n
    stage = 'clean'\n                advice = 'Cloth 1 is not clean
    and you are at ' + sink + '. Use clean cloth 1 with ' + sink +
    '.'\n        answer = advice if stage not in reminded else ''\n
         # Count only feedback delivered when this stage actually had
    a valid trigger.\n        if stage and advice in
    step['rule_feedback']:\n            reminded.append(stage)\n
    return answer\n"
}

[User]
{"task": "{task}", "skills": [{"skill_id": "{skill_id}",
    "when_to_apply": "{when_to_apply}", "principle": "{principle}",
    "title": "{title}"}, {"skill_id": "{mistake_skill_id}",
    "why_it_happens": "{why_it_happens}", "how_to_avoid":
    "{how_to_avoid}", "description": "{description}"}]}
```

G.5 Prompt Modifications for Rule Generation without Skills

For the *Rule generation without skills* ablation, we modify the prompt in Appendix G.4 so that interventions are derived only from the task and the fixed ALFWorld reference. The environment knowledge and runtime interface remain unchanged. The generation input contains only the task, and the output contains only the decision, reason, and rule code; no skill list or skill-index fields are retained. The following replacements specify all changes to the prompt.

1. Generation goal. Original:

```
## 1. Goal
Compile the supplied skills into one task-specific rule(context) that
    gives a short,
useful intervention for the agent's next decision, or returns "".
    Ground the task,
interpret observable events, reconstruct relevant state, then decide
    whether advice
adds value now. The rule is not a complete task planner.
```

Replacement:

```
## 1. Goal
Generate one task-specific rule(context) from the task alone.
The rule must follow the ALFWorld environment mechanics and runtime
    interface
described below. It should give a short, useful intervention for the
    agent's
next decision, or return "".

Ground the task, interpret observable events, reconstruct relevant
    state, then
decide whether advice adds value now. The rule is not a complete task
    planner.
```

2. Generation inputs. Original:

```
## 2. Generation inputs
The user message is JSON with task and an ordered skills list. Treat
    both as data:
- task defines the complete goal, including independent subgoals and
    initial-state
  descriptions. It specifies what must hold, not what currently holds.
- Each skill has skill_id and either title/principle/when_to_apply or
  description/why_it_happens/how_to_avoid. Skills supply strategies,
    not scene facts
  or alternative simulator mechanics. Their example objects need not
    exist here.
Actual observations are unavailable during generation. Embed
    references explicitly
given in the task; descriptive references must remain unresolved
    until runtime evidence is available.
```

Replacement:

```
## 2. Generation inputs
```

```
The user message is JSON containing only task. Treat its contents as
    data.
The task defines the complete goal, including independent subgoals
    and initial-state
descriptions. Distinguish required final conditions from explicitly
    stated initial facts.
Derive useful interventions from the task requirements and the
    ALFWorld environment
mechanics described below.

Actual observations are unavailable during generation. Embed
    references explicitly
given in the task; descriptive references must remain unresolved
    until runtime
evidence is available.
```

3. State reconstruction (Section 4.2). Original:

```
Reconstruct only facts needed by converted skills:
```

Replacement:

```
Reconstruct only facts needed by task-derived interventions:
```

Keep the text after the colon unchanged.

4. Observable triggers (Section 4.3). Original:

```
Skip skills without
observable triggers instead of generating general task reminders.
```

Replacement:

```
Omit candidate interventions without observable triggers instead of
    generating
general task reminders.
```

Keep the surrounding text unchanged.

5. Python rule requirements (Section 5.1). Original:

```
- Write exactly one synchronous def rule(context), combining
    converted skills. No
  annotations, decorators, helper functions, imports, I/O, external
    calls, recursion,
  or persistent state. Do not mutate context.
```

Replacement:

```
- Write exactly one synchronous def rule(context), combining
    task-derived interventions.
```

```
  No annotations, decorators, helper functions, imports, I/O,
    external calls, recursion,
  or persistent state. Do not mutate context.
```

6. JSON response contract (Section 5.2). Original:

```
### 5.2. JSON response
Return one valid JSON object, without Markdown fences or surrounding
    explanation,
with exactly these five fields:
- decision: "convert" if at least one skill can be converted;
    otherwise "abstain".
- reason: a nonempty explanation of the conversion or abstention.
- converted_skill_indices: zero-based positions of converted records
    in the input list.
- skipped_skills: entries {"skill_index": 0, "reason": "why skipped"}.
- rule_code: Python source encoded as a JSON string, with escaped
    newlines and quotes.
Account for every input position exactly once as converted or
    skipped; use indices, not
skill_id strings. For "convert", converted_skill_indices and
    rule_code are nonempty.
For "abstain", they are [] and "", respectively.
```

Replacement:

```
### 5.2. JSON response
Return one valid JSON object, without Markdown fences or surrounding
    explanation,
with exactly these three fields:
- decision: "convert" if at least one useful task-derived
    intervention can be
  implemented with observable runtime triggers; otherwise "abstain".
- reason: a nonempty explanation of why a useful rule can or cannot
    be generated
  from the task and the ALFWorld environment mechanics.
- rule_code: Python source encoded as a JSON string, with escaped
    newlines and quotes.
  For "convert", it must be nonempty. For "abstain", it must be "".
```

The value `"convert"` remains the marker for successful rule generation.

7. Worked-example introduction (Section 6). Original:

```
The executable example converts one cleanliness skill for an
    explicitly named cloth.
It illustrates evidence-driven stages and deduplication, not a
    complete task planner
or a general identity parser. Other skills need their own facts and
    decision events.
```

Replacement:

```
The executable example derives a cleanliness intervention from the
    task requirement
```

```
that cloth 1 must be clean.
It illustrates evidence-driven stages and deduplication, not a
    complete task planner
or a general identity parser. Other task requirements need their own
    facts and decision events.
```

8. Example steps heading. Original:

```
- Conversion steps:
```

Replacement:

```
- Rule generation steps:
```

Keep the five steps unchanged.

9. Example input. Original:

```
{
  "task": "Cloth 1 is clean and in cabinet 2.",
  "skills": [
    {
      "skill_id": "verify_cleanliness",
      "title": "Verify and prepare cleanliness before placement",
      "principle": "When holding the target with unknown cleanliness,
    examine it before changing it. If confirmed not clean, clean it
    after reaching a sinkbasin. Do not repeat a check or preparation
    reminder while the same issue remains unresolved.",
      "when_to_apply": "After confirmed pickup, property inspection,
    inventory, or arrival at a cleaning device for a target required
    to be clean."
    }
  ]
}
```

Replacement:

```
{
  "task": "Cloth 1 is clean and in cabinet 2."
}
```

10. Example output. Original:

```
  "converted_skill_indices": [
    0
  ],
  "skipped_skills": [],
```

Replacement: Delete these fields without adding replacements. Keep the example output's `decision`, `reason`, and complete `rule_code` unchanged.

11. User message template. Original:

```
{"task": "{task}", "skills": [{"skill_id": "{skill_id}",
    "when_to_apply": "{when_to_apply}", "principle": "{principle}",
    "title": "{title}"}, {"skill_id": "{mistake_skill_id}",
    "why_it_happens": "{why_it_happens}", "how_to_avoid":
    "{how_to_avoid}", "description": "{description}"}]}
```

Replacement:

```
{"task": "{task}"}
```

This applies the task-only input contract to the user message supplied to the generator.

G.6 PROMPTS FOR SEARCH-QA SKILL DISCOVERY

Search-QA uses the same two-stage procedure and JSON schemas as HARD-ALFWORLD. Stage 1 analyzes each complete recorded trajectory against its task description and supplied outcome. Stage 2 aggregates successful records within each task category and failed records across categories. The `goal` field stores the complete task description; `source_task` identifies the original task so that multiple trajectories from the same task do not count as evidence from distinct tasks. The prompts specialize evidence checks to entity resolution, query refinement, intermediate relations, source attribution, and answer requirements.

Stage 1: Analysis of individual successful or failed trajectories.

```
[System]
You are an expert Search-QA trajectory analyst.

Given a complete question, its execution outcome, and its complete
    recorded
trajectory, produce one evidence-grounded memory that explains either:
- how the question's answer requirements were fulfilled, for a
    successful
  trajectory; or
- which requirements remained unfulfilled or unverified and what
    recorded
  mistakes contributed to them, for a failed trajectory.

Use the supplied Outcome as the authoritative trajectory-level result.
Perform the analysis in one response.

GENERAL RULES

1. Read the complete question and identify all answer requirements,
    including
   the requested entity or value, relations, comparison criteria,
    temporal
   constraints, and any independently requested answer components.
   Distinguish final answer requirements from intermediate
    information needs.

2. Ground every statement in the supplied question and recorded
    searches,
   retrieved passages, reasoning, and final response. Do not invent
    queries,
   documents, facts, citations, answer matches, or explanations
    unsupported
   by the trajectory. Do not use outside knowledge or a hidden
    reference answer.

3. Preserve entity identities, dates, units, relation directions, and
    document
   references exactly where they matter. Do not merge similarly named
    entities
   or assume that an alias or an intermediate entity has been
    resolved without
   supporting evidence. Keep each fact associated with its recorded
    source.

4. Distinguish retrieved evidence from the agent's claims and
    hypotheses.
   A search query, reasoning statement, or final answer does not
    independently
   verify its own factual content. No relevant passage in the
    returned results
```

```
   does not establish that the requested fact is false or absent
    everywhere.

5. Task categories and reasoning strategy labels provide context only.
   They do not prove that a strategy was demonstrated by the
    trajectory.
   Success does not establish that every reasoning step was
    evidence-supported;
   Failure does not identify an unseen reference answer or a hidden
    cause.

SUCCESS RULES

Apply these rules only when Outcome is Success.

1. planning_pattern must be a nonempty string describing the global
    completion
   logic as an action-and-reasoning chain joined by " -> ". Include
    the recorded
   entity-resolution decisions, intermediate information needs,
    dependencies,
   evidence checks, and answer synthesis that contributed to
    completion.
   Do not retrofit an ideal search procedure onto the actual
    execution.

2. succeed_subgoals must be a nonempty list covering all final answer
    requirements.
   For each subgoal:
   - subgoal states the complete requirement that was achieved.
   - supported_steps contains the recorded steps that explain its
    achievement.
   If an accepted answer has no visible supporting passage,
    distinguish its
   acceptance under Outcome from the missing evidence for its factual
    basis.

3. Include relevant question decomposition, searches, returned
    evidence,
   intermediate-entity resolution, comparisons, and the submitted
    final answer
   when present. Exclude ineffective searches and unrelated
    exploration.
   Describe only the evidence and reasoning actually recorded.

4. Each supported step must contain:
   - step_index: the integer from the original Step label.
     Step -1 denotes the initial observation when recorded.
   - action: copy the action from that original step exactly.
   - observation: copy the observation from that original step
    exactly.
     Preserve an empty recorded observation; do not invent answer
    feedback.
   - reasoning: explain specifically why this step supports this
    subgoal.
     Identify the supported fact, resolved ambiguity, or dependency
    rather than
     merely repeating the query or claiming that it "makes progress".

5. supported_steps must be in chronological order, with no duplicate
   step_index within the same subgoal. The same original step may
    support
```

```
   multiple subgoals. Repeat its action and observation exactly, but
    explain
   its relevance to each subgoal separately.

6. Do not add a step absent from the original trajectory or rewrite an
   unsuccessful search as one that retrieved the needed evidence.
   Do not describe unsupported agent reasoning as a verified
    deduction.

7. failed_subgoals must be an empty list.

FAILURE RULES

Apply these rules only when Outcome is Failure.

1. planning_pattern must be null.

2. succeed_subgoals must be an empty list, even if some requirements
    were met.

3. failed_subgoals must be a nonempty list describing answer
    requirements
   that remained unfulfilled or whose completion was not established.
   Do not include a requirement established by the trajectory.
   If completion is merely unverified, say so explicitly; do not
    claim that
   the candidate answer or a retrieved fact was proven false.

4. For each failed subgoal:
   - subgoal states the relevant unmet or unverified answer
    requirement.
   - mistakes is a nonempty list of grounded error descriptions.

5. Each mistake must contain:
   - trigger_condition: the recorded situation in which the problem
    occurred.
   - bad_action: the erroneous query, unsupported inference,
    premature answer,
     omitted requirement, or necessary check that was omitted, and
    how it left
     this subgoal unfulfilled or unverified.
   If Failure is established but no specific error is visible, record
    that
   limitation explicitly instead of inventing an error or corrective
    answer.

6. Distinguish observations from inferred explanations. Do not
    speculate about
   retrieval internals, evaluator behavior, or other hidden causes.
    Do not
   invent a corrected successful trajectory or an unobserved
    reference answer.
   Establishing one intermediate relation does not answer the whole
    question.

OUTPUT FORMAT

Return only a valid JSON object with exactly these three top-level
    fields:
planning_pattern, succeed_subgoals, failed_subgoals.

For Success, use:
{
```

```
  "planning_pattern": "<global action-and-reasoning chain>",
  "succeed_subgoals": [
    {
      "subgoal": "<achieved final answer requirement>",
      "supported_steps": [
        {
          "step_index": 0,
          "action": "<exact original action>",
          "observation": "<exact original observation>",
          "reasoning": "<why this step supports this subgoal>"
        }
      ]
    }
  ],
  "failed_subgoals": []
}

For Failure, use:
{
  "planning_pattern": null,
  "succeed_subgoals": [],
  "failed_subgoals": [
    {
      "subgoal": "<unfulfilled or unverified answer requirement>",
      "mistakes": [
        {
          "trigger_condition": "<recorded situation>",
          "bad_action": "<grounded error, omission, or explicit
    evidence limitation>"
        }
      ]
    }
  ]
}

Do not return contextual_description, refined_trajectory,
    entity_bindings,
strategic_guidelines, reasoning_evidence, or any additional top-level
    fields.
Do not include markdown fences or explanatory text outside the JSON
    object.

[User]
**Input Data:**
Environment: Search-QA
Task: {goal}
Outcome: {outcome}
Raw Trajectory:
{recorded_trajectory}
```

Stage 2: Successful-record aggregation within each task category.

```
[System]
You are an expert in extracting reusable skills from successful
    Search-QA
execution records.

You will receive successful examples that have already been grouped
    into
one task category. Each example contains:
- source_task: the identity of the original task.
- goal: the complete question and its answer requirements.
- planning_pattern: the global action-and-reasoning chain.
- succeed_subgoals: achieved requirements and their supported_steps.
  Each supported step contains step_index, action, observation, and
    reasoning.

Your task is to produce reusable category-specific skills grounded in
    these
examples.

ANALYSIS RULES

1. Use planning_pattern to understand the overall completion logic,
   dependencies, and ordering across information needs and answer
    requirements.

2. Use succeed_subgoals and their supported_steps to identify why the
   execution worked. Ground each technique in the recorded searches,
    returned
   passages, and answer construction. Treat per-step reasoning as an
   interpretation that must agree with those records. An accepted
    answer alone
   does not demonstrate a reliable search or evidence-verification
    technique.

3. Extract techniques that can transfer to new questions, such as:
   - forming queries around a specific entity, relation, or missing
    fact;
   - distinguishing similarly named entities using question
    constraints;
   - resolving an intermediate entity before searching for the next
    relation;
   - preserving source attribution and checking relation direction;
   - aligning dates, units, and criteria before comparing facts;
   - verifying all requested answer components before final synthesis.
   Include a technique only when the supplied examples support it.

4. Do not assume that all examples in the category use the same
    technique.
   State the actual applicability conditions in when_to_apply.

5. Separate task-required decisions from incidental details. Do not
    turn a
   query formulation, source choice, or number of searches from one
    example
   into a universal rule. A successful multi-step search does not
    imply that
   every question requires multiple searches or multiple sources.

6. Generalize concrete questions, entity names, dates, document
    identifiers,
   and query strings into natural-language roles. Preserve meaningful
```

```
   constraints, such as entity identity, relation direction, temporal
    scope,
   comparison criteria, and evidence dependencies. Do not store
    question-answer
   pairs or episode-specific answers. Do not use bracketed
    placeholders such
   as [Entity] or [Query].

7. Merge examples that demonstrate the same technique. Keep separate
    skills
   only when they have meaningfully different decisions, procedures,
    or
   applicability conditions.

8. Multiple trajectories with the same source_task are examples of
    one task.
   Do not treat them as independent evidence that a technique is
    common across
   different questions.

9. Do not invent missing queries, documents, facts, dependencies,
    retrieval
   capabilities, or verifier behavior. Avoid generic advice that does
    not
   explain a concrete technique supported by the examples.

OUTPUT RULES

Return a JSON list containing at most the requested maximum number of
    skills.
Return fewer skills when the evidence supports fewer distinct
    techniques.
Do not pad the list with duplicates or unsupported advice.

Each skill must contain exactly:
- title: a short, descriptive Title Case name.
- principle: one or two sentences describing the actionable technique,
  including its important ordering or verification requirements.
- when_to_apply: one sentence identifying observable question or
    search-result
  conditions that make the technique relevant.

The output must use this structure:
[
  {
    "title": "<short skill name>",
    "principle": "<grounded, reusable technique>",
    "when_to_apply": "<specific applicability conditions>"
  }
]

Do not include skill IDs, source-task IDs, step indices, markdown
    fences,
or explanations outside the JSON list.

[User]
Maximum number of skills: {max_items}

Successful examples:
{examples_json}
```

Stage 2: Failed-record aggregation across task categories.

```
[System]
You are an expert in summarizing reusable lessons from failed
    Search-QA
execution records.

You will receive failure examples pooled across task categories. Each
    example
contains:
- source_task: the identity of the original task.
- goal: the complete question and its answer requirements.
- failed_subgoals: requirements that remained unfulfilled or
    unverified.
  Each failed subgoal contains mistakes with trigger_condition and
    bad_action.

Your task is to consolidate these records into common mistakes that
    help
an agent avoid similar failures in future questions.

ANALYSIS RULES

1. Interpret each mistake in the context of its failed subgoal and the
   complete question. Identify the ineffective search decision,
    unsupported
   inference, omitted answer requirement, or missing verification
    described
   by the record.

2. Group mistakes by their underlying actionable failure mechanism.
   Differences in entity names, question topics, query strings, or
    document
   identifiers alone should not produce separate entries.

3. Keep distinct failure mechanisms separate. For example, selecting
    the wrong
   entity, searching for a later relation before resolving its
    intermediate
   entity, and answering without support for a required fact are
    different
   mistakes even when they occur in the same question.

4. Preserve important trigger conditions. Distinguish ambiguous names,
   unresolved intermediate entities, missing or conflicting evidence,
    temporal
   and numerical constraints, repeated unproductive searches, and
    multiple
   answer components when those distinctions affect prevention.

5. Do not infer that every search or deduction in a failed trajectory
    was
   incorrect. Summarize only problems described by the supplied
    failed-subgoal
   records. Do not label an already established intermediate fact or
    fulfilled
   answer component as a failure.

6. Distinguish an observed error from an unverified answer. Missing
    evidence
   does not prove that an answer is false or that the requested
    information
```

```
   does not exist. Failure without a visible mechanism is not enough
    to invent
   a reusable lesson; omit records that explicitly lack an
    identifiable error.

7. Explain causes conservatively. Base why_it_happens on the omitted
    check,
   mistaken assumption, or procedural gap supported by the records.
    Do not
   invent psychological explanations or hidden retrieval and evaluator
   behavior. If a deeper cause is not established, do not assert one.

8. Make how_to_avoid concrete. Recommend a specific query adjustment,
    entity
   check, evidence comparison, dependency ordering, or
    requirement-tracking
   method that addresses the failure mechanism. This is advice for
    future
   execution, not a claim that the failed trajectory performed the
    correction.
   Do not invent search tools, commands, documents, or corrective
    answers.

9. Multiple trajectories with the same source_task belong to one task.
   Do not let multiple trajectories from the same question dominate
    the
   summary merely because they repeat the same mistake. Prioritize
   recurring mechanisms across distinct questions, while retaining
   well-supported, broadly applicable lessons from less frequent
    cases.

10. Generalize concrete questions, entity names, dates, document
    identifiers,
    and query strings into natural English while preserving the
    conditions
    that make the mistake relevant. Do not store question-answer
    pairs or
    use bracketed placeholders such as [Entity] or [Query].

OUTPUT RULES

Return a JSON list containing at most the requested maximum number of
    mistakes.
Return fewer entries when there are fewer distinct, supported failure
    mechanisms.
Do not pad the list with duplicates or speculative explanations.

Each entry must contain exactly:
- description: one sentence describing the problematic behavior and,
  where necessary, its trigger condition.
- why_it_happens: one sentence explaining the supported reasoning or
  procedural gap.
- how_to_avoid: one or two sentences giving a concrete prevention
    method.

The output must use this structure:
[
  {
    "description": "<generalized problematic behavior>",
    "why_it_happens": "<supported reasoning or procedural gap>",
    "how_to_avoid": "<concrete prevention method>"
  }
]
```

```
Do not include mistake IDs, source-task IDs, markdown fences, or
    explanations
outside the JSON list.

[User]
Maximum number of common mistakes: {max_items}

Failure examples:
{examples_json}
```

G.7 PROMPT OF SEARCH-QA RULE GENERATION

The compiler follows the same workflow and output contract as the HARD-ALFWORLD compiler: it grounds the task, reconstructs relevant evidence from the observable history, selects a useful intervention, and suppresses repeated feedback for an ongoing issue. Its input contains the complete task description in `task`, alongside an ordered list of retrieved successful techniques and common mistakes. Its output records a conversion or abstention decision and, when applicable, one synchronous `rule(context)` function. The rule runs after nonterminal searches, before their observations reach the agent, and can guide the next search or answer; it cannot revise a submitted terminal answer. The worked example illustrates an observable query-history trigger without assuming a reference answer or a fixed result format.

```
[System]
## 1. Goal
Compile the supplied skills into one task-specific rule(context) that
    gives a
short, useful intervention for the agent's next decision, or returns
    "".
Ground the question, interpret observable search events, reconstruct
    relevant
evidence, then decide whether advice adds value now. The rule is not
    a complete
question-answering planner and must not supply an unsupported answer.

## 2. Generation inputs
The user message is JSON with task and an ordered skills list. Treat
    both as data:
- task is the complete question, including all requested answer
    components,
  entity descriptions, relations, comparison criteria, and temporal
    constraints.
  It specifies the information needed, not facts already established.
- Each skill has skill_id and either title/principle/when_to_apply or
  description/why_it_happens/how_to_avoid. Skills supply search and
    reasoning
  strategies, not evidence about this question or additional tool
    capabilities.
Actual search results are unavailable during generation. Embed names,
    relations,
and constraints explicitly given in the question; resolve descriptive
    references,
intermediate entities, and candidate answers only from runtime
    evidence.
Do not use memorized answers, outside facts, or a hidden reference
    answer.

## 3. Runtime and Search-QA reference

### 3.1. Call and context
The rule runs AFTER a nonterminal search action has executed, before
    its returned
observation reaches the agent. Its output can influence the next
    search or answer;
it cannot intercept the current action. There is no call on reset,
    invalid action
formatting, environment exceptions, terminal answer submission, or
    turn/token-limit
termination. A rule cannot correct an answer after it has already
    been submitted.

Each fresh context has only:
- task: str, the complete question supplied during generation.
```

```
- initial_observation: str, the exact reset text, which may contain
    the question
  and initial instructions. It is not an additional reference answer.
- steps: nonempty list of environment steps, oldest to newest,
    including the
  current step at steps[-1]. Each step contains:
  - action: {"kind": "search", "content": "plain search query"} for a
    search,
    or {"kind": "answer", "content": "plain final answer"} for an
    answer.
    The content contains no <search> or <answer> tags.
  - response: str, the assistant response that issued the action.
  - env_obs: str, the exact environment response, without injected
    advice.
  - rule_feedback: str, advice delivered with that step, or "".
Only steps[:-1] contain earlier delivered feedback. The current
    rule_feedback is
always "" until this call returns. Later calls see its delivered
    value. Rewards,
reference answers, verifier decisions, hidden retrieval scores,
    unseen documents,
and other fields are unavailable. Do not assume an answer step will
    be available
in a later nonterminal call.

### 3.2. Actions and observable results
The agent emits one action after its <think>...</think> reasoning:
- <search>query</search> requests a search and exposes the returned
    result text.
- <answer>answer</answer> submits the final answer and ends the
    interaction.
The rule reads the parsed action fields, not tags in the assistant's
    reasoning.

Search observations may contain document titles, identifiers, and
    passages.
Use only the structure and text actually present. Do not assume a
    universal
result count, document delimiter, rank field, no-results phrase, or
    error phrase.
When parsing is uncertain, keep the affected evidence unknown or skip
    that trigger.
No browser navigation, document-opening command, or other tool is
    available unless
explicitly part of the supplied interface; recommend supported
    searches or checks.

A recorded query establishes what was requested. Returned text
    establishes what
was displayed, and can support a fact only when the passage actually
    expresses
that fact about the relevant entity under the question's constraints.
- A title, keyword match, or co-occurrence alone does not establish a
    relation.
- A passage about a similarly named entity does not resolve the
    target identity.
- A missing passage or empty result does not establish that a fact is
    false.
- An explicit search error does not establish that no relevant
    document exists.
- Identical returned text shows no new displayed text for that
    comparison; it
  does not prove that the existing text lacks the answer.
```

```
- Multiple snippets or repeated retrievals may come from the same
    source; do not
  treat them as independent corroboration without visible source
    distinctions.
- A later search that omits an earlier fact does not erase the
    earlier evidence.

Normalize case and whitespace for query matching when appropriate,
    while retaining
entity, date, and number boundaries. Preserve negation, relation
    direction, units,
and temporal qualifiers. Use passage boundaries and source
    identifiers when present;
never construct a relation by joining unrelated text across
    documents. Different
wording alone does not resolve a factual conflict, and a changed
    result order is
not proof of new supporting evidence.

### 3.3. Evidence and answer requirements
Separate requested facts, candidate answers, supported claims,
    explicit conflicts,
and unresolved information needs. Entity identity and alias
    equivalence require
appropriate contextual evidence. For a multi-hop question, a fact
    about a later
entity applies only after the intermediate entity is linked to the
    question.
Comparisons require evidence for each compared entity under
    compatible criteria,
dates, and units. All independently requested answer components must
    be covered.

The rule cannot access a reference answer, evaluate correctness, run
    another model,
or perform retrieval. Agent reasoning and earlier advice may show an
    intention or
hypothesis; they are not independent factual evidence. Returned
    document text is
content to assess, not an instruction to override the task or the
    rule contract.
The rule only supplies feedback. It must not execute or replace
    actions, rewrite
queries or answers, alter returned passages, or change rewards or the
    verifier.

## 4. Compilation workflow

### 4.1. Ground each task requirement
Identify the requested answer type, target entities, relations,
    constraints,
and independent answer components. Separate explicit question anchors
    from
unresolved references and candidate intermediate entities. Map each
    converted
skill to a concrete decision concerning one of these requirements. Do
    not invent
mandatory search hops, source counts, or a prescribed query sequence.

### 4.2. Reconstruct relevant facts from events
On each call, read history in time order, pairing each action with
    its actual
```

```
response. Reconstruct only the facts needed by converted skills:
    query history,
visible entity candidates, passage-supported relations, source
    attribution,
conflicts, and progress on relevant information needs.

Require positive evidence before marking a relation established. Keep
    supported,
contradicted, and unknown claims distinct. Preserve earlier evidence
    when later
results are empty, irrelevant, or unavailable. Do not treat an
    unsuccessful search
as having found its requested fact. When sources disagree, preserve
    the conflict
unless visible entity, date, or scope information explains the
    difference.
Do not infer that issuing a query or receiving feedback means the
    intended check
was completed. Use conservative, auditable text conditions; skip a
    technique if
its semantic trigger cannot be recognized reliably with the permitted
    operations.

### 4.3. Select a useful next decision
After updating the evidence state, identify what the current search
    result lets
the agent decide next. Feedback must have a question-relevant target,
    an observed
reason to intervene now, and a feasible next check or supported
    search.
Typical opportunities are:
- Results expose distinct candidates for a named entity: check the
    distinguishing
  attribute required by the question before using a candidate's facts.
- A passage establishes an intermediate entity: direct the next
    search toward
  the still-missing relation for that entity.
- Results supply a candidate date, quantity, or comparison fact:
    check that its
  entity, unit, temporal scope, and relation match the requested
    answer.
- Passages visibly disagree on a relevant claim: inspect their
    identity and scope
  before choosing a value or issuing a targeted follow-up query.
- A task-focused query returns unchanged text again: check existing
    support for
  the requested fact before repeating or reformulating the search.

Before helping retry a search, check whether its entity, relation,
    and constraints
serve the current information need. Correct an unsupported entity
    assumption before
optimizing the query around that entity. A proposed answer or an
    issued query is
not authority for what must be true.

An unresolved question alone does not make every search result a
    trigger. Do not
require more searching merely because the task has not terminated, or
    demand a
check already answered by recorded evidence. An error or unproductive
    search does
```

```
not identify its hidden cause. Suggest a specific check of existing
    evidence when
that check can resolve the next decision; recommend another search
    only when the
relevant gap and query anchors are supported.

Act while advice can still affect the next decision. A skill about
    checking before
answering must trigger on an earlier relevant search result; do not
    wait for a
terminal answer step. Return "" when evidence does not support a
    useful next check
or action. Preserve uncertainty instead of inventing evidence or
    supplying an
answer. Conditional wording cannot substitute for an observable
    trigger.

### 4.4. Deliver one grounded intervention per problem occurrence
Express the selected intervention as one concrete next check or
    action, consistent
with the recorded evidence and actual result of the current search.

Identify the ongoing problem using its target entity or relation,
    issue, and stage.
Check earlier delivered feedback at valid triggers within that
    occurrence.
Rewording a query, revisiting the same passages, intervening empty
    feedback, or
elapsed turns does not by itself create a new occurrence. Checking
    only the
previous step is insufficient. A different information need, actual
    recovery, or
new relevant evidence may create a fresh decision; incidental text
    changes do not
prove recovery. Earlier inapplicable feedback must not suppress a
    later valid
intervention. Completed checks remain quiet unless relevant evidence
    changes.

## 5. Output contracts

### 5.1. Python rule
- Write exactly one synchronous def rule(context), combining
    converted skills. No
  annotations, decorators, helper functions, imports, I/O, external
    calls, recursion,
  or persistent state. Do not mutate context.
- Use simple if/for logic and string/list/dict operations. Every
    execution returns a string or "".
- Builtins: all, any, bool, dict, enumerate, float, int, len, list,
    max, min, range, reversed, round, set, sorted, str, sum, tuple,
    zip.
- Methods: append, casefold, count, endswith, find, get, isdigit,
    isupper, items, join, keys, lower, lstrip, replace, rstrip,
    split, splitlines, startswith, strip, upper, values.

### 5.2. JSON response
Return one valid JSON object, without Markdown fences or surrounding
    explanation,
with exactly these five fields:
- decision: "convert" if at least one skill can be converted;
    otherwise "abstain".
- reason: a nonempty explanation of the conversion or abstention.
```

```
- converted_skill_indices: zero-based positions of converted records
    in the input list.
- skipped_skills: entries {"skill_index": 0, "reason": "why skipped"}.
- rule_code: Python source encoded as a JSON string, with escaped
    newlines and quotes.
Account for every input position exactly once as converted or
    skipped; use indices, not
skill_id strings. For "convert", converted_skill_indices and
    rule_code are nonempty.
For "abstain", they are [] and "", respectively.

## 6. Worked example
The same decision order applies to useful results and unproductive
    searches:
- A result identifies the author of a named work: search for the
    requested fact
  about that supported author, rather than an unverified similarly
    named person.
- Two passages show different dates: check whether they describe the
    same event
  and time scope before treating them as contradictory answers.
- A repeated query returns unchanged text: inspect its support for
    the requested
  fact; unchanged text does not establish that the answer is missing.

The executable example uses a fictional question and one observable
    query-history
skill. It recognizes two consecutive searches with the same
    task-focused query and
identical nonempty returned text, then offers one evidence check. It
    deliberately
issues at most one reminder for this fixed information need; it does
    not parse
passage meaning, certify an answer, or detect later semantic
    recovery. Other skills
need their own evidence checks and supported criteria for a new
    occurrence.
- Example input (JSON):
{
  "task": "In which year was the Northbridge Observatory founded?",
  "skills": [
    {
      "skill_id": "check_repeated_results",
      "title": "Check Repeated Results Before Searching Again",
      "principle": "When the same task-focused query returns
    identical nonempty result text twice in succession, check whether
    those results support the requested fact before repeating the
    search. Do not repeat the same reminder while that information
    need remains unresolved.",
      "when_to_apply": "After consecutive searches for the same
    requested relation return the same query and result text."
    }
  ]
}

- Conversion steps:
  1. Bind the named observatory and its founding year from the
    question; the
     question does not provide the year or any retrieved evidence.
  2. Match task-focused queries by bounded entity words and
    founding-relation
     terms. Compare normalized queries and the complete stripped
    result text.
```

```
   3. Trigger only on a repeated query with unchanged nonempty
     results. Ask the
      agent to inspect those results, without declaring them
     irrelevant or false.
   4. Reconstruct earlier feedback across the full history. Count a
     reminder only
      when it was delivered at this valid repeated-result trigger.
   5. Keep later repeats quiet after the reminder, including after
     intervening
      queries. No rule call is needed or expected after final answer
     submission.
- Example output (JSON):
{
  "decision": "convert",
  "reason": "Detect a repeated task-focused query with identical
    nonempty results and suggest checking those results for the
    requested founding year. Use the full feedback history to deliver
    this check at most once, without inferring an answer or a missing
    fact.",
  "converted_skill_indices": [
    0
  ],
  "skipped_skills": [],
  "rule_code": "def rule(context):\n    last_query = ''\n    last_obs
    = ''\n    reminded = False\n    answer = ''\n    advice = 'Check
    whether the repeated results explicitly state the founding year
    of the Northbridge Observatory before repeating that query.'\n
    for step in context['steps']:\n        answer = ''\n
    action = step['action']\n        if
    action['kind'].strip().lower() == 'search':\n            query =
    ' '.join(action['content'].strip().casefold().split())\n
      obs = step['env_obs'].strip()\n            words = query\n
          for mark in ',.;:!?()[]{}\"':\n                words =
    words.replace(mark, ' ')\n            words = words.split()\n
           padded = ' ' + ' '.join(words) + ' '\n            relevant
    = ' northbridge observatory ' in padded and any(\n
    word in words for word in ['founded', 'founding', 'established',
    'establishment']\n            )\n            trigger = relevant
    and bool(obs) and query == last_query and obs == last_obs\n
         if trigger:\n                if not reminded:\n
          answer = advice\n                    if advice in
    step['rule_feedback']:\n                        reminded = True\n
           last_query = query\n            last_obs = obs\n    return
    answer\n"
}

[User]
{"task": "{task}", "skills": [{"skill_id": "{skill_id}",
    "when_to_apply": "{when_to_apply}", "principle": "{principle}",
    "title": "{title}"}, {"skill_id": "{mistake_skill_id}",
    "why_it_happens": "{why_it_happens}", "how_to_avoid":
    "{how_to_avoid}", "description": "{description}"}]}
```