Skill-SD: Skill-Conditioned Self-Distillation for Multi-turn LLM Agents

Hao Wang[*1,5†], **Guozhi Wang**[*5‡], **Han Xiao**[*2], **Yufeng Zhou**[5], **Yue Pan**[5], **Jichao Wang**[1], **Ke Xu**[3], **Yafei Wen**[5], **Xiaohu Ruan**[5], **Xiaoxin Chen**[5], **Honggang Qi**[4§]
[1]Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences
[2]The Chinese University of Hong Kong [3]University of Science and Technology of China
[4]University of Chinese Academy of Sciences [5]vivo AI Lab
wanghao251@mails.ucas.ac.cn hgqi@ucas.ac.cn 11085439@vivo.com
[*]Equal contribution [§]Corresponding author [‡]Project lead [†]Intern at vivo
https://skill-sd.github.io/

Abstract

Reinforcement learning (RL) has been widely used to train LLM agents for multi-turn interactive tasks, but its sample efficiency is severely limited by sparse rewards and long horizons. On-policy self-distillation (OPSD) alleviates this by providing dense token-level supervision from a privileged teacher that has access to ground-truth answers. However, such fixed privileged information cannot capture the diverse valid strategies in agent tasks, and naively combining OPSD with RL often leads to training collapse. To address these limitations, we introduce Skill-SD, a framework that turns the agent's own trajectories into dynamic training-only supervision. Completed trajectories are summarized into compact natural language *skills* that describe successful behaviors, mistakes, and workflows. These skills serve as dynamic privileged information conditioning only the teacher, while the student always acts under the plain task prompt and learns to internalize the guidance through distillation. To stabilize the training, we derive an importance-weighted reverse-KL loss to provide gradient-correct token-level distillation, and dynamically synchronize the teacher with the improving student. Experimental results on agentic benchmarks demonstrate that Skill-SD substantially outperforms the standard RL baseline, improving over vanilla GRPO by 14.0/10.9 points on AppWorld/Sokoban and over vanilla OPD by 42.1/40.6 points.

1 Introduction

Reinforcement learning (RL) has become the dominant paradigm for post-training LLM agents on multi-turn interactive tasks. These agents typically act over ReAct-style trajectories that interleave reasoning with external tool calls [39], operating software through APIs [13, 27], navigating web interfaces [38], and solving sequential planning problems. Group Relative Policy Optimization (GRPO) [22] enables effective policy optimization without a separate value network, and recent work shows that GRPO-based training can produce agents surpassing much larger models [5, 8, 46]. However, RL inherently suffers from sparse, delayed reward signals and high sample complexity. In long-horizon agentic tasks, the reward is often binary (task completed or not), making it extremely sparse and providing nearly negligible guidance about *which* tokens or actions are actually useful.

On-policy distillation (OPD) addresses this gap by providing dense token-level supervision from a privileged teacher. On-policy self-distillation (OPSD) further removes the need for a separate teacher model: the same model serves as both teacher (with privileged information) and student (without), enabling efficient single-model training. Self-Distilled Reasoner [51] matches GRPO performance

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

with 8–12$\times$ fewer tokens; SDPO [12] reaches GRPO-level accuracy 6$\times$ faster; KDRL [36] further shows that combining distillation with policy gradients outperforms either alone. These successes, however, are largely confined to tasks with unique ground-truth answers such as mathematical proofs and code solutions, where a fixed correct answer naturally serves as the teacher's privileged information.

Extending self-distillation to multi-turn agent tasks raises two challenges that prior work has not addressed. **(1) What should the teacher know?** Unlike math problems with a single verifiable answer, agent tasks usually admit diverse valid strategies: an AppWorld task can be solved through different API call sequences, and a Sokoban puzzle can be approached from multiple directions. Conditioning the teacher on any single fixed solution constrains the student's exploration and fails to provide rich guidance. **(2) How to keep training stable?** Naively coupling self-distillation with RL for agents often leads to training collapse. When the teacher generates trajectories (off-policy), its distribution drifts from the student's as training progresses, destabilizing the importance-weighted objective.

We present Skill-SD, a framework that turns the agent's own trajectory history into a dynamic, training-only teacher signal. Skill-SD summarizes completed trajectories into compact natural-language *skills* that capture successful behaviors, mistakes, and high-level workflows. These skills provide reusable strategic guidance rather than a single fixed action sequence, allowing the student to explore its own solutions. During training, these skills condition only the teacher; the student generates on-policy trajectories under the plain task prompt. In this way, the student learns to internalize the skills without introducing retrieval dependence at inference time. In this self-distillation setting, the teacher and the student share the same parameters but are conditioned on different prompts. This prompt discrepancy introduces a distribution mismatch similar to off-policy learning: the student's on-policy tokens are scored under the teacher's skill-augmented distribution. Recent analyses show that naive k_3 differentiation can yield gradient-biased updates even in standard on-policy training [26]. The same issue persists in our cross-prompt setting. To address this, we derive an importance-weighted reverse-KL loss ($\rho \cdot k_3$) that corrects this distribution mismatch, ensuring unbiased token-level gradient updates for the self-distillation objective. Moreover, a frozen teacher inevitably falls behind as the student improves, degrading both the skill-conditioned guidance and the distillation signal. To keep the teacher calibrated, Skill-SD dynamically synchronizes the teacher from the latest student checkpoint, so that the privileged signal co-evolves with the student's improving policy. Our contributions are summarized as follows:

- **Skill as dynamic teacher signal.** We propose using trajectory-derived natural-language skills as dynamic privileged information that conditions only the teacher during self-distillation. This preserves the diversity of valid action paths while allowing the student to internalize the guidance (§3.2).
- **Importance-weighted reverse-KL loss.** We derive an importance-weighted reverse-KL loss for the cross-prompt self-distillation, where teacher and student share parameters but differ in prompt conditioning. This loss corrects the per-token gradient bias of the naive k_3 estimator under distribution mismatch (§3.3, Appendix F).
- **Benefit of dynamic teacher synchronization.** Through systematic ablation, we show that periodically synchronizing the teacher with the improving student improves on-policy performance over a frozen teacher; off-policy teacher-owned rollouts collapse during mid-training (§4.3).

We evaluate Skill-SD on multiple agentic benchmarks. Using Qwen3-4B-Instruct-2507 as the base model, Skill-SD achieves 64.9% accuracy on AppWorld and 62.5% on Sokoban, outperforming vanilla GRPO by 14.0 and 10.9 points, respectively, while also surpassing vanilla OPD by large margins. These results demonstrate the effectiveness of skill-conditioned self-distillation for training LLM agents.

2 Related Work

On-policy distillation and self-distillation for LLMs. Recent self-distillation methods for LLMs [1, 9] differ along two axes. First, some rely on distillation alone [41, 51], whereas others integrate distillation with RL—by converting environment feedback into a dense self-distillation

target [12], unifying KD and RL in a single objective [36], or mitigating their interference through selective imitation [48]. Second, the privileged context is typically *fixed or externally supplied*—ground-truth solutions, offline experience, or frontier-model trajectories as in concurrent work pi-Distill [17]. Concurrent OEL [40] extracts transferable experience from trajectories and consolidates it via context distillation, but uses distillation alone without RL reward signals; OpenClaw-RL [30] combines environment-derived next-state signals with RL for agent training. Skill-SD integrates self-distillation with RL in a joint objective, using analytical skills generated from the agent's own rollouts, and formalizes the cross-prompt importance-weighted KL estimator that this setting requires (Appendix F).

KL-regularized policy optimization and estimator design. A recent insight in the $k_1/k_2/k_3$ KL estimator family [20, 21] is that KL *value* estimation and KL *gradient* optimization are distinct problems: the widely used k_3 estimator provides unbiased value estimates but biased gradients when differentiated directly as a loss [16, 26]. RPG [47] further studies the interaction of importance weighting, KL direction, and clipped updates at scale. These analyses consider the standard same-prompt setting; our cross-prompt configuration, where teacher and student share parameters but differ in prompt conditioning, requires the importance weighting derived in Appendix F.

Reinforcement learning for multi-turn LLM agents. GRPO [7, 22] has become the predominant RL algorithm for LLM training due to its memory efficiency. Adapting RL to multi-turn agent settings introduces new challenges: LOOP [5] adapts PPO for long-horizon tasks without a value network, GiGPO [8] proposes two-level advantage estimation for multi-turn GRPO, RAGEN [31] identifies the "echo trap" instability, and AgentEvolver [45] introduces self-attributing mechanisms for fine-grained credit assignment. Agent-R1 [6], AgentGym-RL [33], and AgentRL [46] scale multi-turn RL across diverse environments with principled environment design [14, 29, 50]. These works improve RL algorithms or infrastructure for agents but do not incorporate self-distillation. Skill-SD complements this line by adding an auxiliary distillation loss that transfers skill-conditioned teacher knowledge alongside RL reward signals.

Experience, reflection, and memory in agents. Reflexion [23] stores self-reflections in episodic memory, ExpeL [49] extracts reusable experience, Agent-R [44] trains agents to reflect via iterative self-training, ECHO [11] rewrites hindsight trajectories, and EvolveR [32] synthesizes trajectory outcomes into strategic principles. UI-Genie [34] iteratively boosts agent performance through a self-improving loop between agent model and reward model, while UI-Mem [15, 35] builds a self-evolving experience memory for online RL. A common approach is to retrieve experience and include it in the policy prompt. In Skill-SD, skills instead condition a teacher whose token-level feedback updates the student through SDL. The student uses the same plain task prompt for both rollout and evaluation, internalizing guidance without retrieving skill text at inference. We compare this teacher-side design with policy-prompting controls in §4.4.

3 Method

3.1 Problem Setup and GRPO Backbone

Figure 1 illustrates the overall Skill-SD pipeline. Each task instance x defines a multi-turn interaction with horizon H. A rollout $\tau = (y_1, \ldots, y_T)$ denotes the sequence of action tokens emitted by the agent while interacting with the environment, where $T \leq H$ counts all action tokens across turns. We use a *completion-rate* reward

$$R(x, \tau) = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}[c_k(\tau) \text{ is satisfied}] \in [0, 1], \tag{1}$$

where $\{c_k\}_{k=1}^{K}$ are task-specific verification criteria: state-based unit tests in AppWorld [27] and boxes-on-target counts in Sokoban (see §4.1 for details). This fine-grained reward preserves partial progress that would be discarded by a binary success signal.

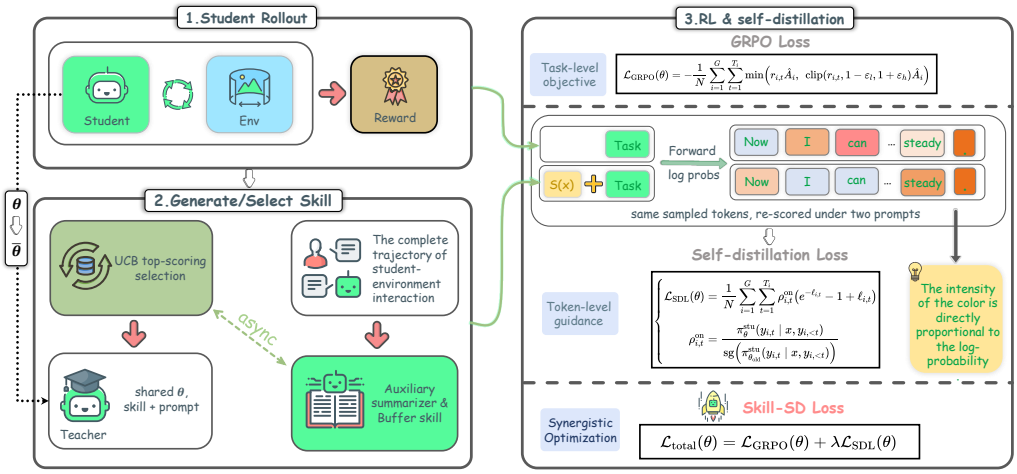

Figure 1: Skill-SD overview. (1) The student generates on-policy rollouts and receives task-level rewards. (2) Completed trajectories are asynchronously summarized into compact skills and stored in a UCB-indexed buffer. (3) The same token sequence is re-scored under the student prompt and the skill-augmented teacher prompt; GRPO provides task-level credit assignment while the importance-weighted SDL loss transfers token-level teacher knowledge. Token color intensity reflects prediction probability.

For each task, we sample a group of G trajectories from the student policy and normalize rewards within the group:

$$\hat{A}_i = \frac{R_i - \frac{1}{G}\sum_{j=1}^{G} R_j}{\mathrm{std}(R_1, \ldots, R_G) + \epsilon}, \tag{2}$$

where $R_i = R(x, \tau_i)$. The GRPO objective is:

$$\mathcal{L}_{\mathrm{GRPO}}(\theta) = -\frac{1}{N}\sum_{i=1}^{G}\sum_{t=1}^{T_i} \min\Big(r_{i,t}\hat{A}_i,\ \mathrm{clip}(r_{i,t}, 1-\varepsilon_l, 1+\varepsilon_h)\hat{A}_i\Big), \tag{3}$$

with student trust-region ratio

$$r_{i,t} = \frac{\pi_\theta^{\mathrm{stu}}(y_{i,t} \mid x, y_{i,<t})}{\mathrm{sg}\big(\pi_{\theta_{\mathrm{old}}}^{\mathrm{stu}}(y_{i,t} \mid x, y_{i,<t})\big)}, \tag{4}$$

and $N = \sum_{i=1}^{G} T_i$ is the total number of valid action tokens in the group, implementing `token-mean` reduction where both long and short trajectories contribute in proportion to their valid-token counts. Following DAPO [43], we use asymmetric clipping bounds with $\varepsilon_h > \varepsilon_l$ to prevent premature entropy collapse.

3.2 Skill-Conditioned Teacher and Dynamic Self-Evolution

Unlike math or code generation, multi-turn agent tasks rarely have a unique ground-truth solution: many action sequences can solve the same task. This rules out fixed-reference distillation but leaves room for a softer form of privileged information: structured summaries of what worked and what failed in past attempts. Conditioning only the teacher on these summaries keeps the student under the same plain prompt during training and evaluation, while transferring teacher guidance through distillation.

The teacher in Skill-SD is therefore *skill-conditioned*: it receives the same task prompt augmented with compact natural-language skills extracted from prior trajectories. Each skill summarizes three

aspects of an attempt:

$$e = \mathrm{Analyze}(\tau, x) = \big(e_{\text{success}}, e_{\text{mistake}}, e_{\text{workflow}}\big), \tag{5}$$

capturing what worked, what failed, and what high-level workflow should be followed next time. We do not store full trajectories as skills. Multi-turn tasks admit many valid action sequences, and a single canonical trace would overconstrain exploration.

Given task x, the student acts on the plain prompt, while the teacher sees the prompt augmented with retrieved skills:

$$\pi_\theta^{\text{stu}}(\cdot \mid x, y_{<t}) = \pi_\theta(\cdot \mid x, y_{<t}), \tag{6}$$

$$\pi_{\bar{\theta}}^{\text{tea}}(\cdot \mid x, S(x), y_{<t}) = \pi_{\bar{\theta}}(\cdot \mid x \oplus S(x), y_{<t}). \tag{7}$$

Here $S(x)$ denotes the retrieved skills and $\bar{\theta}$ is the teacher parameter state. In the **dynamic** setting, $\bar{\theta}$ is synchronized from the latest student checkpoint at each iteration; in the **frozen** setting, $\bar{\theta}$ is fixed throughout training.

Skill retrieval is lightweight. For each task, we select the single highest-scoring skill using a UCB criterion:

$$\mathrm{score}(e) = \bar{r}(e) + c\sqrt{\frac{\ln N_{\text{ucb}}}{n(e)}}, \tag{8}$$

where $\bar{r}(e)$ is the mean reward of skill e, N_{ucb} is the total number of retrievals for the same task, $n(e)$ is the number of times e has been selected for that task, and c controls the exploration–exploitation tradeoff. Skills with $n(e) = 0$ are selected first, ensuring that newly generated skills are always tried before the UCB score is computed. All statistics are maintained per-task. An auxiliary LLM is used only to summarize trajectories into skills; it does not participate in the optimization objective. This is a task-local form of context distillation [10, 25]: the teacher gets richer context during training, but the student does not receive it at test time.

3.3 Importance-weighted reverse-KL loss

The skill-conditioned teacher provides a richer action distribution than the student, but distilling from it is not straightforward. Student and teacher condition on different prompts, so their log-probability ratio is no longer a standard on-policy quantity. Naively differentiating the k_3 divergence estimator yields biased gradients even in standard on-policy settings [26]; in our cross-prompt regime the same issue persists, so an explicit importance-correction term is needed to restore per-token unbiasedness.

In Skill-SD, trajectories are sampled from the old student under the plain prompt, and the skill-conditioned teacher re-scores those same tokens under the augmented prompt. Specifically, we sample

$$\tau_i \sim \pi_{\theta_{\text{old}}}^{\text{stu}}(\cdot \mid x), \tag{9}$$

while recording the old-student log-probabilities during rollout. We then re-forward the sampled sequence under the current student and under the fixed teacher view $\pi_{\bar{\theta}}^{\text{tea}}$ to obtain $\log \pi_\theta^{\text{stu}}$ and $\log \pi_{\bar{\theta}}^{\text{tea}}$.

For each sampled token, define the current-student to teacher-reference log-ratio

$$\ell_{i,t} = \log \pi_\theta^{\text{stu}}(y_{i,t} \mid x, y_{i,<t}) - \log \pi_{\bar{\theta}}^{\text{tea}}(y_{i,t} \mid x, S(x), y_{i,<t}), \tag{10}$$

and the on-policy importance weight

$$\rho_{i,t}^{\text{on}} = \frac{\pi_\theta^{\text{stu}}(y_{i,t} \mid x, y_{i,<t})}{\mathrm{sg}\big(\pi_{\theta_{\text{old}}}^{\text{stu}}(y_{i,t} \mid x, y_{i,<t})\big)}. \tag{11}$$

We then optimize the auxiliary self-distillation loss

$$\mathcal{L}_{\text{SDL}}(\theta) = \frac{1}{N}\sum_{i=1}^{G}\sum_{t=1}^{T_i} \rho_{i,t}^{\text{on}} \left(e^{-\ell_{i,t}} - 1 + \ell_{i,t}\right). \tag{12}$$

The auxiliary SDL term transfers teacher-side skill knowledge into the student under the correct sampling distribution. The weight $\rho_{i,t}^{\text{on}}$ provides the necessary importance correction, since differentiating bare k_3 directly produces biased gradients [16, 26]. The formal gradient identity is given in Appendix F.

For theoretical completeness and for the ablation study in §4.3, we also derive the importance weight under teacher-owned (off-policy) rollout, where trajectories are sampled from $\pi^{\text{tea}}_{\theta_{\text{old}}}(\cdot \mid x \oplus S(x))$. The same construction yields

$$\rho^{\text{off}}_{i,t} = \frac{\pi^{\text{stu}}_{\theta}(y_{i,t} \mid x, y_{i,<t})}{\text{sg}\big(\pi^{\text{tea}}_{\theta_{\text{old}}}(y_{i,t} \mid x, S(x), y_{i,<t})\big)}, \tag{13}$$

preserving the same gradient-correct interpretation under teacher sampling (Appendix F.3).

Equation (12) is a *sampled-token* reverse-KL objective rather than a full-vocabulary distillation loss, and it uses the same `token-mean` aggregation as the GRPO term. For long-horizon agent traces, sampled-token distillation avoids the memory overhead of storing full-vocabulary logits at every token position and requires only the re-forward pass that is already needed for computing $r_{i,t}$.

3.4 Training Objective and Procedure

GRPO and SDL are complementary: GRPO uses trajectory-level reward to select good rollouts, while SDL uses the skill-conditioned teacher to redistribute probability mass at the token level within those rollouts. Neither signal alone suffices. RL without per-token guidance is sample-inefficient in long-horizon tasks, and distillation without reward grounding can drift toward the teacher even when the teacher is wrong. Combining them lets the reward filter bad trajectories while the teacher accelerates credit assignment on good ones.

The overall training objective combines GRPO reward maximization with the auxiliary SDL term:

$$\mathcal{L}_{\text{total}}(\theta) = \mathcal{L}_{\text{GRPO}}(\theta) + \lambda \mathcal{L}_{\text{SDL}}(\theta), \tag{14}$$

where λ controls the distillation strength. The two terms act at different resolutions: $\mathcal{L}_{\text{GRPO}}$ provides task-level credit assignment through group-relative advantages, determining *which* trajectories should be reinforced, while $\mathcal{L}_{\text{SDL}}$ supplies dense token-level guidance toward the skill-conditioned teacher, shaping *how* probability mass is redistributed along those trajectories. These signals are often complementary, but they can disagree when teacher-preferred actions are not aligned with the reward-improving direction on a given sample. The coefficient λ mediates this trade-off: small λ recovers nearly pure GRPO, while larger λ more strongly constrains the student toward the teacher distribution. Within a given update, Proposition 2 shows that $\nabla_\theta(\rho \cdot k_3) = \rho \cdot s_\theta \cdot \ell_t$, so the SDL gradient is exactly zero on sampled tokens where teacher and student agree ($\ell_t = 0$). In the on-policy regime, where ρ^{on}_t stays near 1 under small PPO-style updates, the SDL term is correspondingly self-damping as student–teacher disagreement shrinks; Appendix B confirms this convergence empirically. This property makes SDL an auxiliary shaping signal rather than the primary learning driver, motivating a small λ in practice. Note that although ρ^{on}_t and the GRPO clipping ratio r_t coincide numerically in the main branch, they serve distinct roles—importance correction for SDL versus student-centered trust region for GRPO—as detailed in Appendix F.4 (Table 4). Algorithm 1 in Appendix G summarizes a single training iteration.

4 Experiments

4.1 Setup

Benchmarks. We evaluate Skill-SD on three multi-turn agent environments: AppWorld, Sokoban, and ALFWorld. They cover API-based tool use, spatial planning, and text-based household tasks, respectively.

AppWorld [27] is a comprehensive multi-app API benchmark with 9 simulated consumer applications and 457 APIs. Tasks require multi-step reasoning, cross-application coordination, and iterative API interactions. Correctness is verified by state-based unit tests that check whether the correct state changes were made without undesired side effects. The official training set contains 105 tasks, of which 90 are publicly available [5, 27]. We train on all 90 available tasks and evaluate on the 57-task development set, with a maximum of $H = 40$ interaction turns.

Sokoban is a classic puzzle-planning benchmark requiring strategic reasoning and long-horizon planning. We use 6×6 rooms with 2 boxes and a maximum of $H = 40$ steps. Levels are procedurally generated using the reverse-play method of `gym-sokoban` [19], where the *search depth* parameter

Table 1: Main results on AppWorld and Sokoban (Qwen3-4B-Instruct-2507). Skill-SD uses on-policy student rollout with dynamic teacher synchronization. Subscripts denote absolute change from the base model.

	AppWorld		Sokoban		Avg.	
Method	**Acc.**	**Comp.**	**Acc.**	**Comp.**	**Acc.**	**Comp.**
Base Model	$8.8_{0.0}$	$39.1_{0.0}$	$12.5_{0.0}$	$32.0_{0.0}$	$10.6_{0.0}$	$35.6_{0.0}$
Vanilla OPD	$22.8_{\uparrow 14.0}$	$59.7_{\uparrow 20.6}$	$21.9_{\uparrow 9.4}$	$37.5_{\uparrow 5.5}$	$22.4_{\uparrow 11.7}$	$48.6_{\uparrow 13.0}$
Vanilla GRPO	$50.9_{\uparrow 42.1}$	$76.3_{\uparrow 37.2}$	$51.6_{\uparrow 39.1}$	$68.8_{\uparrow 36.8}$	$51.2_{\uparrow 40.6}$	$72.5_{\uparrow 36.9}$
Skill-Augmented GRPO	$42.1_{\uparrow 33.3}$	$76.1_{\uparrow 37.0}$	$20.3_{\uparrow 7.8}$	$37.5_{\uparrow 5.5}$	$31.2_{\uparrow 20.6}$	$56.8_{\uparrow 21.2}$
Skill-SD	$\mathbf{64.9}_{\uparrow 56.1}$	$\mathbf{84.9}_{\uparrow 45.8}$	$\mathbf{62.5}_{\uparrow 50.0}$	$\mathbf{71.1}_{\uparrow 39.1}$	$\mathbf{63.7}_{\uparrow 53.1}$	$\mathbf{78.0}_{\uparrow 42.4}$

controls the depth of the reverse DFS and thus the minimum solution complexity. Since task diversity is limited, we adopt a difficulty curriculum [3] to maximize signal extraction from each level: the training set consists of 96 levels where the easiest 30% use search depth 15, the middle 50% use depth 20, and the hardest 20% use depth 25. The test set consists of 64 levels with uniform search depth 25.

ALFWorld [24] provides household tasks that require object manipulation through text actions. We train on its 3,553-game `train` split and evaluate 128 episodes from the disjoint 140-game `valid_seen` split, testing held-out instances within the same six task families. Appendix I details the protocol.

Models and metrics. We use Qwen3-4B-Instruct-2507 [18] across all three environments and Qwen2.5-3B-Instruct on ALFWorld. We report **accuracy** (pass@1) and **completion rate** on the AppWorld 57-task development set and the Sokoban 64-level test set, and **success rate** on ALFWorld. Completion rate is the dense reward used for training (Eq. 1). Training configurations appear in Appendices H and I.

Skill writer. For AppWorld and Sokoban, Seed1.8 [4] summarizes completed trajectories. Skill-SD, Skill-Augmented GRPO, and Vanilla OPD use this same writer; Vanilla GRPO uses no skills. For ALFWorld, a frozen copy of each student backbone writes the skills. The skill writer is distinct from the distillation teacher, which re-scores student tokens and is dynamically synchronized with the training student in Skill-SD.

Baselines. We compare against the following baselines:

- **Vanilla GRPO**: standard GRPO with completion-rate reward, without skill prompting or self-distillation.
- **Skill-Augmented GRPO**: GRPO with skills prepended during training and no SDL loss ($\lambda = 0$). Evaluation uses the plain prompt. In AppWorld and Sokoban, the task-indexed bank has no entries for held-out tasks. This baseline contrasts policy-side skill prompting with teacher-side conditioning through SDL.
- **Vanilla OPD**: on-policy distillation without RL reward. The student generates rollouts under the plain prompt; the reverse-KL divergence between the skill-conditioned frozen teacher and the student at each token serves as the per-token advantage in a policy gradient objective. No environment reward is used ($\mathcal{L}_{\text{GRPO}} = 0$), so learning is driven entirely by the distillation signal.
- **Base model**: the Qwen3-4B-Instruct-2507 checkpoint without agent finetuning.

4.2 Main Results

Table 1 compares Skill-SD against all baselines. Skill-SD outperforms Vanilla GRPO by 14.0 points in accuracy on AppWorld and 10.9 points on Sokoban, while also surpassing Vanilla OPD by large margins. Skill-Augmented GRPO underperforms Vanilla GRPO on both benchmarks (42.1% vs. 50.9% on AppWorld; 20.3% vs. 51.6% on Sokoban). Under this training-only prompting protocol,

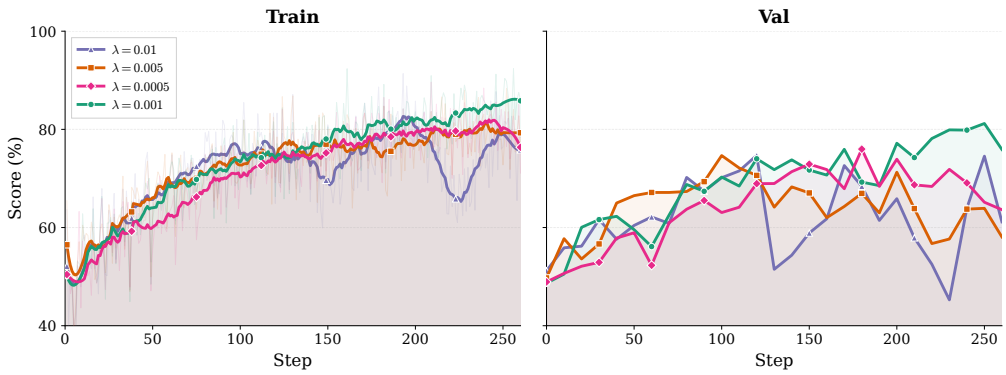

Figure 2: Effect of SDL coefficient λ on AppWorld training (left) and validation (right) completion rate. $\lambda = 0.001$ achieves the best validation performance; $\lambda = 0.01$ over-regularizes and suppresses exploration, while $\lambda = 0.0005$ provides insufficient teacher guidance.

Table 2: Ablation: effect of replacing Skill-SD's design choices. Skill-SD uses on-policy student rollout with dynamic teacher synchronization (bolded). Subscripts denote absolute change from Skill-SD. *Training collapsed during mid-training; values reflect the checkpoint before collapse.

		AppWorld		Sokoban		Avg.	
Rollout	**Teacher**	**Acc.**	**Comp.**	**Acc.**	**Comp.**	**Acc.**	**Comp.**
On-policy	Frozen	49.1$_{\downarrow 15.8}$	79.0$_{\downarrow 5.9}$	50.0$_{\downarrow 12.5}$	63.3$_{\downarrow 7.8}$	49.6$_{\downarrow 14.1}$	71.1$_{\downarrow 6.9}$
On-policy	**Dynamic**	**64.9**$_{0.0}$	**84.9**$_{0.0}$	**62.5**$_{0.0}$	**71.1**$_{0.0}$	**63.7**$_{0.0}$	**78.0**$_{0.0}$
Off-policy*	Frozen	45.6$_{\downarrow 19.3}$	78.8$_{\downarrow 6.1}$	12.5$_{\downarrow 50.0}$	31.3$_{\downarrow 39.8}$	29.1$_{\downarrow 34.6}$	55.0$_{\downarrow 23.0}$
Off-policy*	Dynamic	42.1$_{\downarrow 22.8}$	76.5$_{\downarrow 8.4}$	10.9$_{\downarrow 51.6}$	32.0$_{\downarrow 39.1}$	26.5$_{\downarrow 37.2}$	54.3$_{\downarrow 23.7}$

teacher-side conditioning and distillation provide more effective skill guidance than direct policy prompting.

4.3 Ablation Analysis

We validate Skill-SD's two core design choices—student-owned rollout versus teacher-owned (off-policy) rollout, and dynamic teacher synchronization versus a frozen teacher—by examining what happens when each is replaced. Table 2 reports the effect of each replacement.

Student-owned rollout is essential. Teacher-owned (off-policy) rollout achieves strong early performance but collapses during mid-training in both environments (Figure 3c,d), regardless of whether the teacher is frozen or dynamically refreshed. As the student policy diverges from the teacher's rollout distribution, the importance ratios become increasingly unstable. The collapse is far more severe on Sokoban (12.5%) than on AppWorld (45.6%), consistent with Sokoban's higher task irreversibility.

Dynamic synchronization improves on-policy training. Within the on-policy regime, dynamic teacher synchronization provides consistent gains over a frozen teacher: 15.8 points in accuracy on AppWorld and 12.5 points on Sokoban. The frozen on-policy configuration remains stable but converges to a lower performance plateau.

Teacher-side skills preserve the student prompt distribution. Skill-Augmented GRPO trains $\pi_\theta(a \mid h, e)$ but evaluates $\pi_\theta(a \mid h, \varnothing)$, creating a prompt mismatch as in learning with privileged information [28]. On AppWorld, this baseline achieves a high training score but lower validation accuracy (Figure 5). Skill-SD keeps the student under the plain prompt throughout, transferring skill

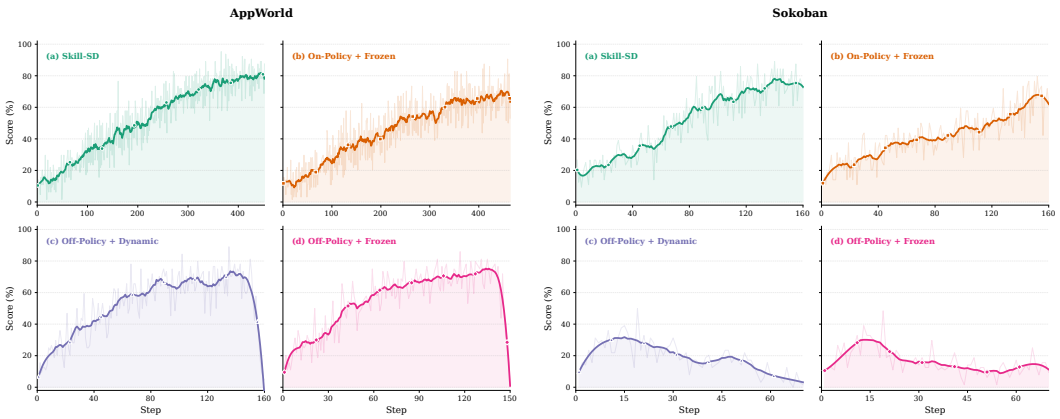

Figure 3: Training dynamics of the four teacher–student configurations on AppWorld (left) and Sokoban (right). Skill-SD (a, on-policy + dynamic) trains stably to high performance. On-policy + frozen (b) converges stably but to a lower plateau. Both off-policy variants (c, d) achieve strong early performance but collapse during mid-training. The pattern replicates across both environments.

guidance through teacher re-scoring and SDL. Appendix D analyzes the prompt mismatch; §4.4 evaluates a complementary baseline that also retains skills at evaluation.

SDL weight λ controls the RL–distillation balance. Figure 2 sweeps $\lambda \in \{0.01, 0.005, 0.001, 0.0005\}$ on AppWorld. $\lambda = 0.001$ achieves the highest validation completion rate; $\lambda = 0.01$ over-regularizes and plateaus, while $\lambda \leq 0.0005$ provides insufficient teacher guidance. The pattern reflects a bias–variance trade-off: larger λ overpowers the RL signal; smaller λ recovers nearly pure GRPO.

4.4 Skill-Writer Capacity and Skill Placement on ALFWorld

We use ALFWorld to study skill-writer capacity, skill placement, and the teacher's privileged context. With a frozen copy of the student as the writer, we compare Skill-SD and GRPO on both backbones using paired seeds 43 and 44. Each pair shares its seed, configuration, and evaluation games. We use $\lambda = 0.001$ for both backbones. Appendix I gives the implementation, controls, and compute measurements.

Table 3: ALFWorld success rates with self-written skills. Each row is a paired run; gains are percentage points computed from the displayed rates.

Backbone	Seed	GRPO	Skill-SD	Gain (pp)
Qwen3-4B-Instruct-2507	43	0.617	**0.758**	+14.1
	44	0.555	**0.695**	+14.0
Qwen2.5-3B-Instruct	43	0.766	**0.852**	+8.6
	44	0.695	**0.828**	+13.3

Table 3 shows gains from self-written skills for both backbones and both seeds. For Qwen3-4B with seed 43, replacing the teacher's compact skills with retrieved successful trajectories gives 0.594 success, while retrieving skills only from other task families gives 0.445, compared with 0.758 for Skill-SD. These comparisons support compact, task-relevant skills as the teacher's privileged context.

To examine skill placement, we compare Skill-SD with two policy-prompting controls using the same self-written skill source. Skill-Augmented GRPO uses skills during training and the plain prompt at evaluation, achieving 0.539 success. Matched skill prompting also retains skills at evaluation, using a frozen training-derived bank with exact-family UCB1 top-1 retrieval, and achieves 0.148. Skill-SD reaches 0.758 while using the plain evaluation prompt. These results favor teacher-side conditioning and distillation over the evaluated policy-prompting protocols. Appendix I examines the skill bank and prompting dynamics.

5 Conclusion

We introduced Skill-SD, a framework combining skill-conditioned self-distillation with GRPO for multi-turn LLM agents. Skills condition only the teacher; the student generates on-policy trajectories under the plain task prompt, gaining per-token supervision without constraining exploration. Dynamic teacher synchronization keeps the distillation signal calibrated, and the importance-weighted SDL loss ensures unbiased gradients (Proposition 2). Skill-SD improves over GRPO across AppWorld, Sokoban, and ALFWorld. The ablations support on-policy student rollouts and dynamic teacher synchronization, while the ALFWorld results show gains with self-written skills across two backbones and paired seeds.

Limitations

Skill-SD uses UCB-based retrieval and a fixed distillation weight; learned retrieval and adaptive weighting may improve robustness. Evaluation at larger model scales and across more training seeds would help characterize scalability and variability. Our analysis establishes token-level gradient properties, leaving trajectory-level guarantees under an evolving teacher open. Beyond sandboxed benchmarks, applications in automation and accessibility require safeguards against harmful or unauthorized actions.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 62271466.

References

[1] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48, 2009. doi: 10.1145/1553374.1553380.

[4] Bytedance Seed. Seed1.8 model card: Towards generalized real-world agency. *arXiv preprint arXiv:2603.20633*, 2026.

[5] Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive LLM agents. *arXiv preprint arXiv:2502.01600*, 2025.

[6] Mingyue Cheng, Jie Ouyang, Shuo Yu, Ruiran Yan, Yucong Luo, Zirui Liu, Daoyu Wang, Qi Liu, and Enhong Chen. Agent-R1: Training powerful LLM agents with end-to-end reinforcement learning. *arXiv preprint arXiv:2511.14460*, 2025.

[7] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

[9] Tommaso Furlanello, Zachary C. Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1607–1616, 2018. URL `https://proceedings.mlr.press/v80/furlanello18a.html`.

[10] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, 2023. doi: 10.18653/v1/2023.findings-acl.507. URL `https://aclanthology.org/2023.findings-acl.507/`.

[11] Michael Y. Hu, Benjamin Van Durme, Jacob Andreas, and Harsh Jhamtani. Sample-efficient online learning in LM agents via hindsight trajectory rewriting. *arXiv preprint arXiv:2510.10304*, 2025.

[12] Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

[13] Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, et al. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*, 2025.

[14] Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Shuai Ren, Hao Wang, Shibo He, and Wenchao Meng. Learnact: Few-shot mobile gui agent with a unified demonstration benchmark. *arXiv preprint arXiv:2504.13805*, 2025.

[15] Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, et al. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments. *arXiv preprint arXiv:2602.06075*, 2026.

[16] Kezhao Liu, Jason Klein Liu, Mingtao Chen, and Yiming Liu. Rethinking KL regularization in RLHF: From value estimation to gradient optimization. *arXiv preprint arXiv:2510.01555*, 2025.

[17] Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

[18] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[19] Max-Philipp Schrader. gym-sokoban: Reinforcement learning environment for the game of sokoban. `https://github.com/mpSchrader/gym-sokoban`, 2018.

[20] John Schulman. Approximating KL divergence. `http://joschu.net/blog/kl-approx.html`, 2020. Blog post.

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

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

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

[24] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=0IOX0YcCdTn`.

[25] Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022. doi: 10.48550/ARXIV.2209.15189.

[26] Yunhao Tang and Rémi Munos. On a few pitfalls in KL divergence gradient estimation for RL. *arXiv preprint arXiv:2506.09477*, 2025.

[27] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 16022–16076, 2024.

[28] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557, 2009. doi: 10.1016/j.neunet.2009.06.042.

[29] Ruiyi Wang and Prithviraj Ammanabrolu. A practitioner's guide to multi-turn agentic reinforcement learning. *arXiv preprint arXiv:2510.01132*, 2025.

[30] Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026.

[31] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Fei-Fei Li, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

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

[33] Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. AgentGym-RL: Training LLM agents for long-horizon decision making through multi-turn reinforcement learning. In *International Conference on Learning Representations*, 2026.

[34] Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, et al. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. *arXiv preprint arXiv:2505.21496*, 2025.

[35] Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, Yafei Wen, and Hongsheng Li. Ui-mem: Self-evolving experience memory for online reinforcement learning in mobile gui agents. *arXiv preprint arXiv:2602.05832*, 2026.

[36] Hongling Xu, Qi Zhu, Heyuan Deng, Jinpeng Li, Lu Hou, Yasheng Wang, Lifeng Shang, Ruifeng Xu, and Fei Mi. KDRL: Post-training reasoning LLMs via unified knowledge distillation and reinforcement learning. *arXiv preprint arXiv:2506.02208*, 2025.

[37] Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. In *Advances in Neural Information Processing Systems*, 2025.

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

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

[40] Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*, 2026.

[41] Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

[42] Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1. In *Advances in Neural Information Processing Systems*, 2024.

[43] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, 2025.

[44] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-R: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.

[45] Yunpeng Zhai et al. AgentEvolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

[46] Hanchen Zhang et al. AgentRL: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. In *International Conference on Learning Representations*, 2026.

[47] Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. On the design of KL-regularized policy gradient algorithms for LLM reasoning. In *International Conference on Learning Representations*, 2026.

[48] Zhaoyang Zhang, Shuli Jiang, Yantao Shen, Yuting Zhang, Dhananjay Ram, Shuo Yang, Zhuowen Tu, Wei Xia, and Stefano Soatto. Reinforcement-aware knowledge distillation for LLM reasoning. *arXiv preprint arXiv:2602.22495*, 2026.

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

[50] Pengxiang Zhao, Guangyi Liu, Yaozhen Liang, Weiqing He, Zhengxi Lu, Yuehao Huang, Yaxuan Guo, Kexin Zhang, Hao Wang, Liang Liu, et al. Mas-bench: A unified benchmark for shortcut-augmented hybrid mobile gui agents. *arXiv preprint arXiv:2509.06477*, 2025.

[51] Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

A Skill Format

Each skill is a structured JSON object with three fields summarizing a completed trajectory. During training, skills are prepended to the teacher prompt to provide task-specific guidance.

Example Skill

success_analysis: "The strategic intent to use specialized apps (Splitwise for identifying roommates, Venmo for sending payment requests) and prioritize authentication for restricted APIs was sound, as these tools are designed to streamline expense splitting and peer-to-peer payments."

mistake_analysis: "Key failures stemmed from unvalidated assumptions (using a hypothetical bill amount instead of retrieving the actual receipt from the file system) and neglecting to verify API parameter requirements upfront, leading to repeated authentication errors."

golden_workflow: "1. Access the file system to retrieve the actual internet bill amount from the stored receipt. 2. Authenticate with a contact app using valid credentials to obtain roommate emails. 3. Authenticate with Venmo, calculate each roommate's equal share, then send payment requests via the correct API."

B Self-Distillation Visualization

Figure 4 visualizes how the student and teacher token distributions evolve during training on a representative AppWorld task. At the start of training, the student and teacher assign notably different probabilities to the same tokens; by the end, their distributions have largely converged. The teacher's distribution remains relatively stable across epochs, while the student progressively learns to match it through the SDL loss. The SDL loss curve (right) decreases by 59.3% over training, confirming this convergence.

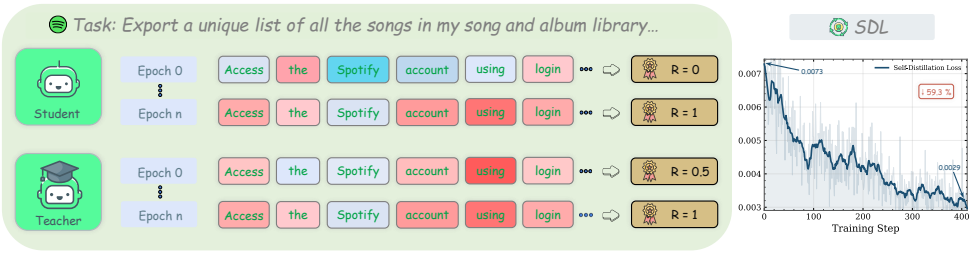

Figure 4: Token-level self-distillation dynamics on an AppWorld task. Token color intensity is proportional to prediction probability. At Epoch 0, student and teacher distributions differ substantially; after convergence, they have largely aligned. Right: the SDL loss decreases by 59.3% over training.

C Trajectory Example and Skill Generation

We present a complete AppWorld interaction trajectory and the skill generated from it. The task requires exporting all Spotify library data to a CSV file, then terminating the account. This trajectory was collected during training and is shown with minimal editing (long API outputs are truncated with `[...]`).

Task: *Export a unique list of all the songs in my song and album library and all playlists in my Spotify account into "~/backups/spotify.csv". The file should have headers "Title" and "Artists" (artists separated by "|"). Terminate my account after this backup is complete.*

Agent – Step 1

To solve this task, we need to: (1) Access the Spotify account using login. (2) Retrieve the full list of songs, albums, and playlists. (3) Extract song titles and artists. (4) Combine into a unique list. (5) Export to CSV. (6) Terminate the account.

```
passwords = apis.supervisor.show_account_passwords()
```

Environment – Step 1


```
[{"account_name": "spotify", "password": "CgR}U-F"}, {"account_name":
"file_system", "password": "+ok}#yD"}, ...]
```


Agent – Step 2

The Spotify password is `"CgR}U-F"`. I will now log in to Spotify.

```
login_result = apis.spotify.login(username='ka_burt@gmail.com',
password=spotify_password)
```

Environment – Step 2


```
{"access_token": "eyJhb...", "token_type": "Bearer"}
```


Agent – Step 3

Login was successful. Now let me check the available Spotify APIs.

```
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
```

Environment – Step 3

`[show_song_library, show_album_library, show_playlist_library, show_song, delete_account, ...]` *[74 API endpoints; truncated]*

Agent – Step 4

The relevant APIs are `show_song_library`, `show_album_library`, `show_playlist_library`, and `show_song`. Let me retrieve the song library first.

```
song_library = apis.spotify.show_song_library(
  access_token=spotify_access_token)
```

Environment – Step 4

Returns 5 songs with titles and artists. E.g., `{"song_id": 202, "title": "Summer's End", "artists": [{"name": "Ethan Wallace"}]}`, ...

Agent – Step 5

Now let me retrieve the album library and playlist library.

```
album_library = apis.spotify.show_album_library(access_token=...)
```

```
playlists = apis.spotify.show_playlist_library(access_token=...)
```

Environment – Step 5

Returns 5 albums (16 song IDs total) and 5 playlists (39 song IDs total).

Agent – Step 6

Now I need to fetch details for each song ID from albums and playlists using `show_song`.

```
song = apis.spotify.show_song(access_token=spotify_access_token,
song_id=song_id)
```

Environment – Step 6: Error

```
Exception: Unexpected parameter 'access_token' passed to the show_song API.
Allowed parameters are: ['song_id']
```

Agent – Step 7

I made a mistake in the API call. Let me check the API specification for `show_song`.

```
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='show_song'))
```

Environment – Step 7

```
{"parameters": [{"name": "song_id", "type": "integer", "required": true}]}
```
— confirms `show_song` only takes `song_id`, no `access_token`.

Agent – Step 8

Confirmed. I will fix the code to call `show_song(song_id=...)` without `access_token`, and collect all songs from song library, albums, and playlists into a deduplicated list.

```
song = apis.spotify.show_song(song_id=song_id) [loops over all song IDs]
```

Environment – Step 8

`59` — 59 unique songs extracted successfully.

Agent – Step 9

59 unique songs collected. Now I need to write the CSV. Let me check the `file_system` APIs and log in.

```
file_system_login = apis.file_system.login(
  password=file_system_password)
```

Environment – Step 9: Error

```
Validation error. Reason: username: Field required
```

Agent – Step 10

The `file_system.login` requires both `username` and `password`. Let me check the API docs.

```
print(apis.api_docs.show_api_doc(app_name='file_system', api_name='login'))
```

Environment – Step 10

Confirms: `username` (string, required) and `password` (string, required).

Agent – Step 11

Now I will log in with both username and password, create the CSV, and write it.

```
apis.file_system.login(username='ka_burt@gmail.com', password=...)
apis.file_system.create_file(file_path='~/backups/spotify.csv',
content=csv_content, ...)
```

Environment – Step 11

```
CSV file created successfully.
```

Agent – Step 12

Backup complete. Now terminate the Spotify account.

```
apis.spotify.delete_account(access_token=spotify_access_token)
```

Environment – Step 12

```
Spotify account terminated successfully.
```

Agent – Step 13

All tasks completed. Marking as done.

```
apis.supervisor.complete_task()
```

Environment – Step 13

```
Execution successful.
```

Generated skill. The following skill is produced from this trajectory by the auxiliary summarizer:

Generated Skill

success_analysis: "The strategic approach of systematically retrieving data from song library, album library, and playlists before combining into a deduplicated export was sound, as was the correct sequencing of backup-then-terminate."

mistake_analysis: "Two API parameter errors occurred: (1) passing `access_token` to `show_song`, which only accepts `song_id`; (2) omitting `username` when calling

`file_system.login`. Both stemmed from not consulting API specifications before the first call."

golden_workflow: "1. Retrieve credentials via supervisor. 2. Log in to Spotify and collect song/album/playlist libraries. 3. For each song ID, call `show_song(song_id=...)` without extra parameters. 4. Log in to file system with both username and password, then write deduplicated CSV. 5. Delete Spotify account and mark task complete."

D Training–Evaluation Prompt Mismatch

A natural baseline is to inject skills directly into the GRPO rollout prompt, training a single policy $\pi_\theta(a \mid h, e)$ conditioned on both history h and skill e. Table 1 shows that this *Skill-Augmented GRPO* underperforms Vanilla GRPO on both benchmarks, reducing validation accuracy from 50.9% to 42.1% on AppWorld and from 51.6% to 20.3% on Sokoban.

Empirical signature. Figure 5 shows different training patterns across the two environments. On AppWorld, Skill-Augmented GRPO reaches the highest *training* score of any method ($\sim$80%), yet validation accuracy falls to 42.1%, consistent with a train–evaluation prompt mismatch. On Sokoban, Skill-Augmented GRPO performs poorly in both training and validation, indicating difficulty learning with directly prompted skills in this setting.

Objective mismatch. Let $\pi_\theta^E(a \mid h) := \pi_\theta(a \mid h, e)$ and $\pi_\theta^0(a \mid h) := \pi_\theta(a \mid h, \varnothing)$. Skill-Augmented GRPO optimizes $J_E(\theta) = \mathbb{E}_{\tau\sim\pi_\theta^E}[R(\tau)]$, but evaluation measures $J_0(\theta) = \mathbb{E}_{\tau\sim\pi_\theta^0}[R(\tau)]$. Even with infinite data, maximizing J_E does not in general maximize J_0: the two objectives correspond to different conditional policies that happen to share parameters θ. This mismatch is consistent with the AppWorld pattern, where a strong skill-conditioned training policy does not translate into a strong plain-prompt evaluation policy.

Gradient misalignment. Let g_E and g_0 be the policy gradients of J_E and J_0 at θ. A step $\theta^+ = \theta + \alpha\, g_E$ improves the evaluation objective only to the extent that $g_0^\top g_E > 0$. Since θ is shared, Skill-Augmented GRPO can help through parameter transfer, but the sign of $g_0^\top g_E$ is not guaranteed. This inner product characterizes whether a local update to the skill-conditioned policy also improves the plain-prompt objective.

Connection to privileged information. This failure mode is analogous to learning using privileged information [28]: training-time features unavailable at evaluation create a structural mismatch. Skill-SD addresses this by confining privileged access to a training-only component: skills condition only the teacher during training, while the student (the evaluation-time policy) generates trajectories and is optimized under the plain prompt. The SDL loss then acts as an implicit knowledge-compilation step, reminiscent of System 2→System 1 distillation [42], transferring privileged behavior through gradient-based distillation rather than direct policy conditioning.

E Prompt Templates

The AppWorld system prompt is adapted from the official AppWorld benchmark [27]. The full prompt including all API specifications is available in the benchmark repository.

AppWorld System Prompt (abbreviated)

I am your supervisor and you are a super intelligent AI Assistant whose job is to achieve my day-to-day tasks completely autonomously.

To do this, you will need to interact with app/s (e.g., spotify, venmo etc) using their associated APIs on my behalf. For this you will undertake a *multi-step conversation* using a python REPL environment.

Here are three key APIs that you need to know to get more information

```
# To get a list of apps that are available to you.
print(apis.api_docs.show_app_descriptions())

# To get the list of apis under any app, e.g. spotify
print(apis.api_docs.show_api_descriptions(app_name='spotify'))

# To get the specification of a particular api
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='login'))
```

Each code execution will produce an output that you can use in subsequent calls. Using these APIs, you can now generate code, that I will execute, to solve the task. Let's start with the task

My name is: {main_user.first_name} {main_user.last_name}. My personal email is {main_user.email} and phone number is {main_user.phone_number}.
Task: How many playlists do I have in Spotify?

[*A multi-turn few-shot demonstration follows, showing authentication, API discovery, pagination, and calling* `apis.supervisor.complete_task()`. *See the AppWorld benchmark repository for the full prompt.*]

Key instructions:
(1) Make sure to end code blocks with ``` followed by a newline.
(2) Remember you can use the variables in your code in subsequent code blocks.
(3) Always look at API specifications before calling an API.
(4) Write small chunks of code and only one chunk per step.
(5) Many APIs return items in "pages". Loop over `page_index`.
(6) Once you have completed the task, call `apis.supervisor.complete_task()`.

My name is: {first_name} {last_name}. My personal email is {email} and phone number is {phone}.
Task: {input_str}

Sokoban System Prompt

You are an expert agent operating in the Sokoban environment.

Symbols and Their Meaning
- Walls (`#`): These block movement. You can't move through or push anything into walls.
- Floor (`_`): Open spaces where you can walk and move boxes.
- Targets (`O`): The spots where boxes need to go.
- Boxes (`X`): These are what you need to push onto the targets.
- Player (`P`): That's you! You'll move around the grid to push boxes.
- Box on Target (`T`): A box successfully placed on a target.
- Player on Target (`S`): You standing on a target.

Your Goal
Your goal is to push all the boxes (`X`) onto the target spots (`O`). Once all boxes are on the targets, you win!

Rules
1. You can only move up, down, left, or right. And don't overthink it.
2. You can only push boxes. You can't pull them, so plan ahead to avoid getting stuck.
3. You can't walk through or push boxes into walls (`#`).
4. To avoid traps, do not push boxes into corners or against walls where they can't be moved again.

You will be provided the current map observation.
After your thought process, you MUST output your final single action enclosed in ``` ```.
Valid Actions: `up` | `down` | `left` | `right`

Skill Generation Prompt

Trajectory Analysis and Golden Skill Generation

You are an expert trajectory analyst. Your task is to analyze the given trajectory and generate a structured golden skill that captures key learnings.

Conversation History
`{conversation_history}`

Trajectory Outcome
- **Result**: `{result_status}`
- **Total Steps**: `{step_count}`
- **Completion Rate**: `{completion_rate}`

Output Requirements (JSON)

You need to deeply analyze the Trajectory Outcome and Conversation History I provided, thinking step by step, and then summarize the skill to guide the model in future encounters.

Output a unique JSON object containing exactly these 3 fields:

1. `success_analysis`: A comprehensive paragraph summarizing the **strategic intent** that led (or would lead) to success.
2. `mistake_analysis`: A comprehensive paragraph analyzing the **root causes** of failures or inefficiencies observed in the trajectory.
3. `golden_workflow`: A concise description of the **ideal overall process** to complete this task from start to finish.

REMEMBER

The analysis must be concise. As an expert, the skill you summarize will guide the model's subsequent operations. Therefore, it's crucial to adopt a guiding approach, focusing primarily on the steps that confuse the model.
Your success-to-failure output ratio should follow the Completion Rate (`{completion_rate}`). If the Completion Rate is high, you should focus on summarizing the success analysis; otherwise, you should focus on summarizing the mistake analysis.

F Importance-Weighted SDL under Student and Teacher Rollouts

This appendix formalizes the sampled-token self-distillation term used in Eq. (12). The analysis follows the modern distinction between *KL value estimation* and *KL gradient optimization*: a bare k_3 estimator can be a good value estimator while still producing the wrong gradient when differentiated directly [16, 20, 26]. That distinction matters here because the auxiliary loss is interpreted as self-evolution. If its gradient is biased under the actual sampling policy, then the loss may still move the model, but it no longer cleanly implements the intended teacher-to-student transfer direction.

F.1 Estimator family

Let q_θ denote the current student distribution, p a fixed teacher reference distribution, and μ the actual sampling distribution at a fixed token position. Define

$$\ell = \log \frac{q_\theta(y)}{p(y)}. \tag{15}$$

The standard reverse-KL estimator family includes

$$k_1(\ell) = \ell, \tag{16}$$

$$k_2(\ell) = \tfrac{1}{2}\ell^2, \tag{17}$$

$$k_3(\ell) = e^{-\ell} - 1 + \ell. \tag{18}$$

Among these, k_3 is attractive because it is non-negative and is widely used as a low-variance reverse-KL estimator [20]. But recent analyses show that differentiating k_3 directly does not in general yield the desired reverse-KL gradient [16, 26].

F.2 General sampled-token SDL

Definition 1 (Importance-weighted SDL)**.** For a fixed prefix, let $y_t \sim \mu(\cdot \mid y_{<t})$. Define

$$\ell_t \triangleq \log q_\theta(y_t \mid y_{<t}) - \log p(y_t \mid y_{<t}), \tag{19}$$

$$\rho_t^\mu \triangleq \frac{q_\theta(y_t \mid y_{<t})}{\mathrm{sg}(\mu(y_t \mid y_{<t}))}, \tag{20}$$

where $\mathrm{sg}(\cdot)$ denotes stop-gradient.

The choice of ρ_t^μ is structurally forced. Its denominator must match the actual sampling distribution μ, otherwise the importance correction does not close under the sampling expectation. Its numerator must be the trainable student, otherwise the product rule would mix fixed-reference terms with the wrong score function and the resulting gradient would not reduce to the intended reverse-KL form.

Proposition 2 (Per-token SDL gradient identity)**.** *Under Definition 1,*

$$\nabla_\theta\big(\rho_t^\mu \cdot k_3(\ell_t)\big) = \rho_t^\mu \cdot s_\theta(y_t) \cdot \ell_t, \tag{21}$$

where $s_\theta(y_t) = \nabla_\theta \log q_\theta(y_t \mid y_{<t})$. Moreover, for any fixed prefix $y_{<t}$,

$$\mathbb{E}_{y_t \sim \mu(\cdot \mid y_{<t})}\big[\nabla_\theta\big(\rho_t^\mu \cdot k_3(\ell_t)\big)\big] = \nabla_\theta D_{\mathrm{KL}}\big(q_\theta(\cdot \mid y_{<t}) \,\|\, p(\cdot \mid y_{<t})\big). \tag{22}$$

Proof. We apply the product rule:

$$\nabla_\theta(\rho_t^\mu \cdot k_3) = (\nabla_\theta \rho_t^\mu) \cdot k_3 + \rho_t^\mu \cdot (\nabla_\theta k_3). \tag{23}$$

Because the denominator of ρ_t^μ is frozen under stop-gradient,

$$\nabla_\theta \rho_t^\mu = \rho_t^\mu \cdot s_\theta(y_t). \tag{24}$$

Since $\log p(y_t \mid y_{<t})$ is computed at the pre-update parameters and does not depend on the current θ, $\nabla_\theta \ell_t = s_\theta(y_t)$. Using $k_3'(\ell) = 1 - e^{-\ell}$, we obtain

$$\nabla_\theta k_3(\ell_t) = (1 - e^{-\ell_t}) \cdot s_\theta(y_t). \tag{25}$$

Substituting both expressions gives

$$\nabla_\theta(\rho_t^\mu \cdot k_3) = \rho_t^\mu \cdot s_\theta(y_t) \cdot k_3(\ell_t) + \rho_t^\mu \cdot (1 - e^{-\ell_t}) \cdot s_\theta(y_t) \tag{26}$$

$$= \rho_t^\mu \cdot s_\theta(y_t) \left[k_3(\ell_t) + 1 - e^{-\ell_t}\right]. \tag{27}$$

Using $k_3(\ell_t) = e^{-\ell_t} - 1 + \ell_t$, the bracket simplifies to ℓ_t, proving Eq. (21). Taking expectation under the sampling distribution yields

$$\mathbb{E}_{y_t \sim \mu(\cdot \mid y_{<t})}[\nabla_\theta(\rho_t^\mu \cdot k_3)] = \sum_{y_t} \mu(y_t \mid y_{<t}) \frac{q_\theta(y_t \mid y_{<t})}{\mu(y_t \mid y_{<t})} s_\theta(y_t) \ell_t \tag{28}$$

$$= \sum_{y_t} q_\theta(y_t \mid y_{<t}) s_\theta(y_t) \ell_t \tag{29}$$

$$= \nabla_\theta D_{\mathrm{KL}}\big(q_\theta(\cdot \mid y_{<t}) \,\|\, p(\cdot \mid y_{<t})\big), \tag{30}$$

which proves the conditional unbiasedness claim. □

F.3 On-policy and off-policy corollaries

On-policy main method. For the main Skill-SD update, $\mu = \pi^{\text{stu}}_{\theta_{\text{old}}}(\cdot \mid x, y_{<t})$ and $p = \pi^{\text{tea}}_{\bar{\theta}}(\cdot \mid x, S(x), y_{<t})$. Therefore

$$\rho_t^{\text{on}} = \frac{\pi^{\text{stu}}_\theta(y_t \mid x, y_{<t})}{\mathrm{sg}\big(\pi^{\text{stu}}_{\theta_{\text{old}}}(y_t \mid x, y_{<t})\big)} \tag{31}$$

is the importance weight for the main self-evolution signal. Numerically, ρ_t^{on} equals the GRPO ratio r_t, but the role is different: ρ_t^{on} weights the auxiliary reverse-KL term, whereas r_t defines the clipped reinforcement-learning surrogate.

Off-policy comparison branch. For the teacher-rollout comparison branch, $\mu = \pi^{\text{tea}}_{\theta_{\text{old}}}(\cdot \mid x, S(x), y_{<t})$, which yields

$$\rho_t^{\text{off}} = \frac{\pi^{\text{stu}}_{\theta}(y_t \mid x, y_{<t})}{\text{sg}\big(\pi^{\text{tea}}_{\theta_{\text{old}}}(y_t \mid x, S(x), y_{<t})\big)}. \tag{32}$$

This is the off-policy correction emphasized in RPG-style analyses [47]: without it, the differentiated k_3 term would not follow the intended reverse-KL transfer direction under teacher sampling.

F.4 Why SDL and GRPO need distinct interpretations

Skill-SD uses two different quantities for two different jobs:

- The main-branch SDL weight $\rho_t^{\text{on}} = \pi^{\text{stu}}_{\theta}/\pi^{\text{stu}}_{\theta_{\text{old}}}$ must match the *student sampling distribution*.
- The off-policy SDL weight $\rho_t^{\text{off}} = \pi^{\text{stu}}_{\theta}/\pi^{\text{tea}}_{\theta_{\text{old}}}$ must match the *teacher sampling distribution*.
- The GRPO clipping ratio $r_t = \pi^{\text{stu}}_{\theta}/\pi^{\text{stu}}_{\theta_{\text{old}}}$ must stay centered at the *old student policy*, even when we apply DAPO-style clip-higher asymmetric bounds [43].

If one naively reused the teacher-denominator ratio for clipping, the trust region would no longer be centered at 1 because the teacher and student differ by prompt conditioning. This is exactly the type of mis-centering that recent off-policy analyses warn against [37, 47]. In the main on-policy branch, ρ_t^{on} happens to equal r_t numerically, but the interpretation is still different: the former defines the self-evolution update, whereas the latter defines the clipped RL surrogate.

Table 4: Importance-weighted SDL and student-centered GRPO in Skill-SD.

Quantity	Definition	Purpose
Main SDL weight ρ_t^{on}	$\pi^{\text{stu}}_{\theta} / \text{sg}(\pi^{\text{stu}}_{\theta_{\text{old}}})$	Match student sampling in the main method
Off-policy SDL weight ρ_t^{off}	$\pi^{\text{stu}}_{\theta} / \text{sg}(\pi^{\text{tea}}_{\theta_{\text{old}}})$	Match teacher sampling in the comparison branch
GRPO clipping ratio r_t	$\pi^{\text{stu}}_{\theta} / \text{sg}(\pi^{\text{stu}}_{\theta_{\text{old}}})$	Center trust region at old student

G Experimental Details

Variant definition. Table 5 summarizes the exact axes varied in the current AppWorld study.

Table 5: Variant definitions for the AppWorld ablation matrix.

Variant	Rollout Source	Teacher Refresh	Distillation
Off-policy + dynamic	Teacher	Dynamic self-evolution	Yes
Off-policy + frozen	Teacher	Frozen	Yes
On-policy + dynamic	Student	Dynamic self-evolution	Yes
On-policy + frozen	Student	Frozen	Yes
Skill-Augmented GRPO	Skill-conditioned student	N/A	No
Vanilla GRPO	Student	N/A	No
Vanilla OPD	Teacher/student re-forward pair	Frozen	Yes, no RL
Baseline	Qwen3-4B-Instruct-2507	N/A	No

AppWorld protocol. We use the official AppWorld train/dev split [27]. Accuracy is pass@1 on the dev set. Completion rate is the fraction of task-specific unit tests satisfied by the final environment state.

Loss reduction and clipping. Both $\mathcal{L}_{\text{GRPO}}$ and $\mathcal{L}_{\text{SDL}}$ use `verl`'s `token-mean` reduction: losses are summed over valid action tokens and divided by the total valid-token count. For GRPO, we use DAPO's clip-higher setting [43], i.e., asymmetric clipping with $\varepsilon_h > \varepsilon_l$.

Algorithm 1 Skill-SD training

Require: Student policy π_θ, skill bank $\mathcal{B}$, group size G, SDL coefficient λ, learning rate η

for iteration $= 1, 2, \ldots$ **do**
  $\theta_{\text{old}} \leftarrow \theta$; sync teacher $\bar{\theta} \leftarrow \theta$ ▷ *dynamic*; or keep $\bar{\theta}$ fixed (*frozen*)
  — Rollout —
  for each task x in the batch **do**
    $S(x) \leftarrow \text{UCB-retrieve}(\mathcal{B}(x))$ ▷ Eq. 8; $\emptyset$ if no prior skills
    Sample $\{\tau_i\}_{i=1}^{G} \sim \pi^{\text{stu}}_{\theta_{\text{old}}}(\cdot \mid x)$; record $\log \pi^{\text{stu}}_{\theta_{\text{old}}}$
    Compute rewards $R_i = R(x, \tau_i)$ and advantages $\hat{A}_i$ ▷ Eq. 2
  end for
  — Re-score —
  for each $(x, \{\tau_i\})$ **do**
    Compute $\log \pi^{\text{stu}}_{\theta}(y_{i,t} \mid x,\, y_{i,<t})$ ▷ current student, plain prompt
    Compute $\log \pi^{\text{tea}}_{\bar{\theta}}(y_{i,t} \mid x \oplus S(x),\, y_{i,<t})$ ▷ teacher, augmented prompt
  end for
  — Policy Update —
  $r_{i,t} = \rho^{\text{on}}_{i,t} \leftarrow \pi^{\text{stu}}_{\theta} \big/ \text{sg}\big(\pi^{\text{stu}}_{\theta_{\text{old}}}\big)$ ▷ trust-region ratio & IS weight
  $\ell_{i,t} \leftarrow \log \pi^{\text{stu}}_{\theta} - \log \pi^{\text{tea}}_{\bar{\theta}}$ ▷ student–teacher log-ratio (Eq. 10)
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{GRPO}}(\theta;\, r,\, \hat{A}) \;+\; \lambda\, \mathcal{L}_{\text{SDL}}(\theta;\, \rho^{\text{on}},\, \ell)$ ▷ Eqs. 3, 12
  $\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}$
  Async: summarize new trajectories $\rightarrow$ skills; update $\mathcal{B}$ ▷ consumed next iter.
end for

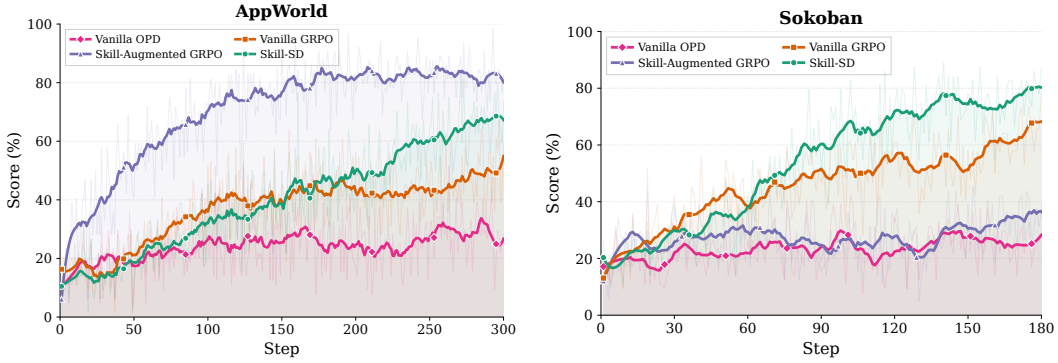

Figure 5: Training curves for all baselines on AppWorld (left) and Sokoban (right). On AppWorld, Skill-Augmented GRPO achieves the highest *training* accuracy while Skill-SD steadily narrows the gap. On Sokoban, Skill-SD overtakes Vanilla GRPO after step 60 and maintains steady gains.

Training curves. Figure 5 shows the training dynamics of all baselines on both benchmarks. On AppWorld, Skill-Augmented GRPO achieves the highest *training* accuracy but overfits severely: its validation accuracy (42.1%, Table 1) falls well below Skill-SD (64.9%). Skill-SD improves steadily on both training and validation throughout. On Sokoban, Skill-SD starts slower than Vanilla GRPO but overtakes it after step 60 and continues to improve steadily. In both environments, Vanilla OPD remains at low performance throughout training.

Training efficiency. Relative to vanilla GRPO, Skill-SD adds teacher re-scoring and trajectory summarization into skills. In a naïve synchronous implementation, these API requests would block the rollout and parameter-update pipeline, substantially increasing wall-clock time per training step. We adopt a fully asynchronous architecture that decouples skill generation from the main training loop: API requests are dispatched in the background and their results are consumed by subsequent iterations rather than the current one. This design overlaps writer latency with rollout and gradient computation. Appendix I reports measured ALFWorld training times and GPU-hours, including teacher re-scoring and the amortized cost of the shared writer.

H Training Hyperparameters

For AppWorld and Sokoban, we implement Skill-SD using the rLLM framework[1] with the verl backend[2]. Trajectory summarization into skills is performed by Seed1.8 [4]. GRPO follows Shao et al. [22]. Multi-turn rollouts use a *Token-In-Token-Out* (TITO) mode, where the model reads and generates raw tokens rather than going through the chat completion API. This avoids the token-ID inconsistencies that arise when applying chat templates to multi-turn message histories—a discrepancy that can cause the training distribution to diverge from the rollout distribution and destabilize advantage estimation.

Table 6: Training hyperparameters for AppWorld and Sokoban experiments.

Hyperparameter	**Value**
Group size G (rollouts per task)	4
Train batch size	16
Optimization epochs per batch	1
Max prompt length	8192 tokens
Max response length	20480 tokens
Dynamic batch sizing	Enabled
Ulysses sequence parallel size	4
Max tokens per GPU	14336
Training temperature	1.0
Evaluation temperature	0.0 (greedy)
GPU memory utilization	0.7
Entropy coefficient	0.0
DAPO clip ε_l	0.2
DAPO clip ε_h	0.28
Advantage normalization ε	1×10^{-8}
Learning rate	1×10^{-6}
Weight decay	0.0
UCB exploration coefficient c	$\sqrt{2}$
SDL coefficient λ	0.001
Optimizer	AdamW
GPU	8×NVIDIA L40S

The UCB exploration coefficient $c = \sqrt{2}$ follows the UCB1 algorithm of Auer et al. [2], which is theoretically optimal for rewards bounded in $[0, 1]$ via Hoeffding's inequality, guaranteeing logarithmic regret.

[1] `https://github.com/rllm-org/rllm`

[2] `https://github.com/volcengine/verl`

I ALFWorld Experiments

I.1 Experimental setup

We evaluate skill-writer capacity, explicit skill prompting, and alternative privileged contexts on ALFWorld [24]. We implement the experiments with the open-source SDAR framework[3], open-weight models, and public data. In Skill-SD, the student uses the plain prompt during both training and evaluation; only the teacher receives skills and re-scores the student's sampled tokens.

We train on ALFWorld's 3,553-game `train` split, using 16 environments and eight rollouts per environment (128 episodes per step). We evaluate on 128 episodes from the disjoint 140-game `valid_seen` split. The six task families cover placing one object, placing two objects, examining an object under a light, and heating, cooling, or cleaning an object before placing it. This evaluation tests held-out instances within the same task families.

For each backbone, a frozen copy of the student writes the skills: Qwen3-4B-Instruct-2507 writes for the 4B student, and Qwen2.5-3B-Instruct writes for the 3B student. This writer is distinct from the dynamically synchronized teacher used for token-level self-distillation. We use $\lambda = 10^{-3}$ for both backbones. Paired comparisons use the same seed and evaluation games, with identical initial policies. The main comparisons use seeds 43 and 44; the remaining controls use seed 43 unless otherwise stated.

I.2 Self-written skills and writer capacity

With self-written skills, Skill-SD achieves higher success than GRPO for both seeds and both backbones (Table 7). Replacing the 3B self-writer with DeepSeek-V4-Flash increases success for seed 43 but decreases it for seed 44. The stronger writer gives mixed results across the two seeds.

Table 7: ALFWorld success rate (↑) for paired seeds. The DeepSeek-V4-Flash column changes only the skill writer for the 3B student; dashes denote settings not evaluated.

Student	Seed	GRPO	Skill-SD, self-writer	Skill-SD, DeepSeek writer
Qwen3-4B	43	0.617	**0.758**	—
Qwen3-4B	44	0.555	**0.695**	—
Qwen2.5-3B	43	0.766	0.852	**0.891**
Qwen2.5-3B	44	0.695	**0.828**	0.758

I.3 Skill placement and alternative privileged contexts

We compare teacher-side conditioning with two explicit-prompting controls that use the same self-written skill source. Skill-Augmented GRPO inserts skills into the policy prompt during training and uses the plain prompt at evaluation. Matched skill prompting retains skills at evaluation, using a bank constructed exclusively from training trajectories. The bank is frozen during evaluation: evaluation trajectories, rewards, and labels neither create nor update entries. For this matched control, retrieval exact-matches the task family and uses UCB1 for top-1 skill selection.

Teacher-side conditioning with SDL outperforms both explicit-prompting controls (Table 8). The comparison with Skill-Augmented GRPO evaluates skill placement and the SDL objective together under a shared skill source.

Compact skills outperform retrieved successful trajectories as teacher context under the same SDL objective. The trajectory control reaches 0.594, compared with 0.758 for Skill-SD. Restricting skill retrieval to other task families gives success 0.445, with a measured same-family retrieval rate of 0.000 throughout training. These comparisons favor task-relevant compact skills in the tested retrieval scheme.

Diagnostics for matched skill prompting. Prompt and response clipping ratios are both 0.000 in the matched-prompting run. Adding a skill increases the prompt from approximately 495 to 825

[3] `https://github.com/ZJU-REAL/SDAR`

Table 8: Qwen3-4B controls on ALFWorld, seed 43. All arms use the same 128 held-out evaluation games. Success is a rate; higher is better.

Training configuration	Skills at evaluation?	Success ↑
Vanilla GRPO	No	0.617
Skill-SD	**No**	**0.758**
Skill-Augmented GRPO	No	0.539
Matched skill prompting	Yes	0.148
SDL with a retrieved successful trajectory	No	0.594
SDL with cross-family skills only	No	0.445

tokens. Skills summarize both successful and failed trajectories. At step 150, the matched arm's bank contains 19,200 entries, of which 81.6% summarize failed trajectories. Family-level matching can retain instance-specific details: for example, a skill for examining an object under a light names `sidetable 1` and `desklamp 1`. Such details can be inappropriate for a held-out game's objects or layout.

Both prompting arms share the writer, skill format, bank construction, seed, and hyperparameters. At step 150, each bank contains 19,200 entries, with successful-trajectory fractions between 18.34% and 18.43%. Both start at success 0.203 with empty banks. Matched prompting performs worse at all 15 evaluation checkpoints, ending at 0.148 versus 0.539. Context interference from retained instance-specific skill text offers a possible explanation.

Training dynamics of direct prompting. At step 150 in ALFWorld, Skill-Augmented GRPO has policy entropy 0.724 versus 0.336 for Skill-SD, and generates approximately 626k versus 441k tokens per batch.

I.4 Sensitivity to the distillation weight

The SDL coefficient affects success in the seed-43 Qwen3-4B setting (Table 9), where we vary only this coefficient. The value $\lambda = 10^{-3}$ gives the highest success among those tested, while 10^{-2} leads to training collapse. The paired runs for both backbones use 10^{-3} without retuning. Neighboring settings perform below the GRPO control.

Table 9: SDL-coefficient sensitivity on ALFWorld, Qwen3-4B, seed 43. The paired GRPO success rate is 0.617.

λ	3×10^{-4}	10^{-3}	3×10^{-3}	10^{-2}
Success ↑	0.508	**0.758**	0.562	0.000

I.5 Training cost

We measure the seed-43 Qwen3-4B pair on eight H200 GPUs per run (Table 10). Per-step times average 135 pure training steps. Teacher re-scoring takes 23.3 seconds, or 7.5% of a Skill-SD step: a forward pass over student-sampled tokens without teacher rollout or environment interaction.

An asynchronous four-GPU vLLM writer serves six concurrent runs, an amortized $4/6$ GPU per run; results are collected in a subsequent training step. With this shared allocation, wall-clock time increases by 4.9% and end-to-end GPU-hours by 13.7%. A dedicated writer uses more GPU-hours.

Table 10: Measured training cost on ALFWorld, Qwen3-4B, seed 43. Amortized GPU-hours include the four-GPU writer shared across six runs.

Quantity	GRPO	Skill-SD	Increase
Seconds per pure training step	294.5	310.2	5.3%
Total training wall-clock (h)	13.22	13.87	4.9%
Amortized end-to-end GPU-hours	105.76	120.21	13.7%

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction describe teacher-side skill conditioning, its integration with GRPO, and dynamic teacher synchronization (Sections 1 and 3). Section 4 and Appendix I report results on AppWorld, Sokoban, and ALFWorld with the corresponding evaluation protocols.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The Limitations section discusses UCB-based retrieval, fixed distillation weights, evaluation scale and seed coverage, token-level theoretical guarantees, and deployment safeguards. Appendix I reports the coefficient sweep and prompting diagnostics.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: Appendix F states the fixed-prefix and fixed-reference assumptions and provides a complete proof of Proposition 2, with on-policy and off-policy corollaries.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [No]

   Justification: The paper provides the objective, Algorithm 1, prompt templates (Appendix E), AppWorld and Sokoban training hyperparameters (Appendix H), and ALFWorld protocols (Appendix I). A complete configuration and executable reproduction package for the ALFWorld implementation are not included, so full reproduction of all reported results is not yet supported.

   Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [No]

   Justification: The benchmarks and base models are publicly available, but the paper does not provide a public release of the complete training and evaluation code with commands and environment specifications for reproducing all results.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [No]

   Justification: Section 4.1 and Appendices G and H report the AppWorld and Sokoban setup and training hyperparameters. Appendix I specifies the ALFWorld data splits, rollout counts, skill writers, controls, and seeds, but does not provide the complete ALFWorld optimizer and generation configuration.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

Justification: The AppWorld and Sokoban configurations are single-run results; Appendix I reports paired results for two seeds in the main ALFWorld comparisons, while the remaining controls use one seed. Individual seed outcomes are reported, but they do not establish statistical significance or characterize variance across all experiments.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [No]

   Justification: Appendix H reports eight NVIDIA L40S GPUs for AppWorld and Sokoban; Appendix I reports eight H200 GPUs per ALFWorld training run, a four-GPU writer shared across six runs, and measured wall-clock times and amortized GPU-hours for one paired comparison. Complete per-experiment resource accounting and the total compute used by the full research project, including preliminary or failed runs, are not reported.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The study uses public benchmarks and publicly released base models (Section 4.1 and Appendices H and I), does not involve human subjects, and discusses scope boundaries and deployment risks in the Limitations section.

Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: The Limitations section identifies automation and accessibility as potential applications of more capable agents and calls for safeguards against harmful or unauthorized actions when deploying beyond sandboxed benchmarks.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper presents a training method evaluated on existing public models and sandboxed benchmarks. No new pretrained frontier model or scraped dataset is distributed with this manuscript.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [No]

    Justification: The cited assets include the following: AppWorld, Qwen3, rLLM, and verl use Apache-2.0 licenses, gym-sokoban uses an MIT license, and Seed1.8 is accessed under its provider's API terms. Licenses and terms for all ALFWorld assets and skill writers are not fully documented in this version.

    Guidelines:

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

13. **New assets**

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

    Answer: [N/A]

    Justification: No new datasets, model checkpoints, or code packages are distributed with this manuscript.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: This paper does not involve research with human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: Section 3.2 and Appendix H describe the auxiliary Seed1.8 skill summarizer used for AppWorld and Sokoban. Appendix I describes the ALFWorld frozen self-writers, the DeepSeek-V4-Flash writer control, and their distinction from the synchronized distillation teacher.

    Guidelines:

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