HARLO: HIERARCHICAL AGENTIC RL WITH INFORMATION-GAIN GUIDED CREDIT ASSIGNMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic reinforcement learning (RL) has advanced LLM-based agents, yet most existing algorithms are confined to single-agent settings. The Planner-Executor architecture, where a planner generates high-level plans and an executor executes step-by-step actions, offers a promising paradigm for such tasks, but efficient and stable RL methods for co-optimizing both agents remain lacking. In this paper, we propose Harlo, a hierarchical agentic RL framework that addresses the credit assignment problem through an information-theoretic approach. Our key insight is that information gain—measured as the behavioral influence of the plan on the executor's actions via counterfactual evaluation—serves as a natural signal for allocating credit. This signal enables a two-level credit assignment strategy that dynamically reshapes advantage values for both agents, ensuring stable joint optimization and co-evolution. Evaluated on ALFWorld, ScienceWorld and WebShop, Harlo consistently outperforms strong and diverse baselines. Code is available at https://anonymous.4open.science/r/Harlo-F2F5.

1 INTRODUCTION

Agentic reinforcement learning (RL), represented by GRPO (Shao et al., 2024), has significantly advanced the training of LLM-based agents on complex interactive tasks. However, most existing algorithms are designed for single-agent settings. In reality, many complex, long-horizon tasks—such as robotic collaboration (Feng et al., 2025), software development (Qian et al., 2024), and scientific experimentation (Ghareeb et al., 2026)—naturally demand multi-agent cooperation. Among these, the Planner-Executor architecture is a classic and efficient paradigm: the planner generates high-level plans, and the executor interacts with the environment step by step following the plan. While the Planner-Executor architecture has shown promising results on several tasks, there still lacks efficient and stable RL methods to consistently improve its performance.

We formalize this co-learning problem as Hierarchical Agentic RL, where the planner and the executor need to learn distinct but cooperative behaviors from shared outcome rewards. This entails three challenges: (1) **Agent-level credit assignment.** The final outcome reward does not directly indicate the contributions of each agent, making the learning directions of each agent confused. (2) **Step-level credit assignment.** Even when agent-level credit is properly assigned, long-horizon tasks involve numerous decision steps, and it remains unclear which steps contribute most. (3) **Non-stationarity during joint optimization.** Since both the planner and executor are updated during training, the contribution of each agent depends on the capacity of each other. Without an adaptive credit assignment mechanism, the joint optimization may easily fall into local optima.

Existing RL-based methods for optimizing the Planner-Executor policies fall into two categories. One extends single-agent algorithms like GRPO to multi-agent settings by fixing one agent while training the other (Si et al., 2026; Zhao et al., 2026). While these approaches avoid the difficulties brought by joint training, they can hardly achieve co-evolving of the whole system, which largely limits their performance. Other lines of work propose some joint training methods for multi-agent settings. However, they either equally broadcast the credit or split the credit by a fixed ratio. (Mo et al., 2025; Liu et al., 2026b) Consequently, the joint optimization process becomes highly inefficient and unstable. Overall, while existing methods successfully apply RL to optimize planner and executor policies, the credit assignment problem still remains largely underexplored.

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

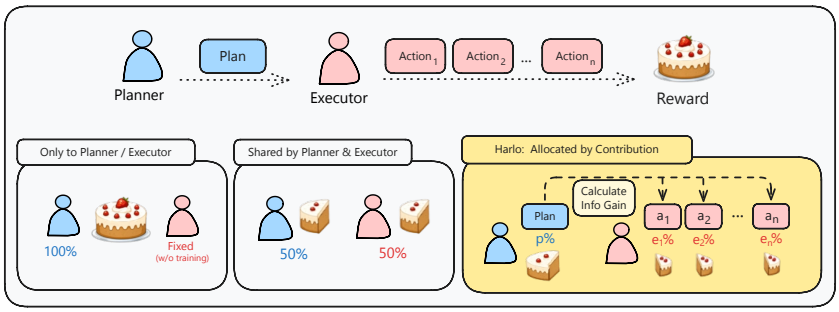


Figure 1: The credit assignment challenge in the Planner–Executor architecture. **Top:** The planner produces a plan that guides the executor's step-by-step interaction, after which the environment returns a single outcome reward. **Bottom:** Existing paradigms either grant all credit to one role while freezing the other (left), or share credit uniformly across roles (middle). Harlo (right) instead estimates each role's actual contribution via information gain and allocates credit to the plan and to each individual action proportionally.

To address these challenges, we propose **Harlo**, a hierarchical agentic reinforcement learning framework for the Planner–Executor architecture. The key idea is to leverage **information gain**—the behavioral influence between execution with and without planning—as a real-time signal for fine-grained credit assignment. Unlike naive credit assignment driven by prior knowledge, Harlo computes the influence of planning on execution, enabling cross-role credit assignment between the planner and executor. Intuitively, if removing the plan barely changes the executor's behavior, the planner contributes little to the final outcome, and vice versa. Therefore, information gain serves as a natural signal that reflects the extent to which planning influences the executor's actions. Specifically, Harlo follows a plan-then-execute paradigm and computes step-wise information gain through counterfactual evaluation of the plan's influence on actions. Harlo introduces a fine-grained two-level credit assignment mechanism that reshapes the advantage values of both plans and actions. In this way, each agent can attribute its own contribution to the final outcome, allowing both the planner and the executor to improve their capabilities during joint training and achieve co-evolution.

Our main contributions are threefold:

- **Counterfactual information gain as an intrinsic credit assignment signal.** We formulate Planner–Executor co-learning as a hierarchical credit assignment problem and identify counterfactual information gain as an effective signal for measuring how much the planner influences executor behaviors. This provides an annotation-free basis for distinguishing the contributions of planning and execution under sparse outcome supervision.
- **An information-gain-guided hierarchical RL framework.** We propose Harlo, a new hierarchical agentic RL framework for Planner-Executor that performs both agent-level (planner vs. executor) and step-level (within each agent's trajectory) credit assignment. Through information-gain-based credit assignment, Harlo dynamically allocates agent-level credit to the planner and step-level credit to executor decisions, providing differentiated optimization signals for stable and efficient Planner–Executor co-learning and mitigates the non-stationarity problem.
- **Comprehensive evaluation on diverse long-horizon benchmarks.** Experiments on ALFWorld, ScienceWorld, and WebShop with 1.5B and 7B-scale models demonstrate consistent improvements over strong baselines, with particularly significant gains on long-horizon tasks, for example, achieving up to +288% improvement on ScienceWorld L1 over GRPO.

2 Related Works

Hierarchical Architectures for LLM Agents. Early LLM agents interleave reasoning and acting within a single model, as exemplified by ReAct (Yao et al., 2022b). To handle long-horizon tasks, subsequent work decouples high-level planning from low-level execution. Prompt-based methods

refine plans through interaction or recursive decomposition (Wang et al., 2023; Sun et al., 2023; Prasad et al., 2024). Training-based methods instead optimize the planner itself: PLAN-AND-ACT (Erdogan et al., 2025) supervises a dedicated planner with synthesized planning data, EAGLET (Si et al., 2026) trains a plug-and-play global planner while keeping the executor frozen and assesses plan quality by contrasting executor rollouts with and without the plan, Co-EPG (Zhao et al., 2026) alternately trains one module while freezing the other. However, none provides an online estimate of the planner's contribution during joint optimization.

Credit Assignment in Agentic RL. Reinforcement learning has become the dominant paradigm for post-training LLM agents on interactive tasks. Trajectory-level methods such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024) assign a single shared advantage to the entire rollout. To obtain finer granularity, GiGPO (Feng et al., 2026a) groups actions taken from the same anchor state across rollouts and computes step-level relative advantages without extra rollouts, ArCHer (Zhou et al., 2024) trains a turn-level critic whose values guide token-level policy updates in a hierarchical actor–critic scheme, and iStar (Liu et al., 2025) and PilotRL (Lu et al., 2025) pursue similar step- or stage-wise refinements. In parallel, counterfactual and information-theoretic approaches such as CCPO (Li et al., 2026) and IGPO/InfoPO (Wang et al., 2026; Kong et al., 2026) estimate the marginal contribution of each turn to the final outcome and use it as a dense learning signal. However, these methods cannot determine from a shared outcome reward whether a failure stems from a poor plan or poor execution.

Multi-Agent Co-Learning. Training distinct yet interdependent agents has long been studied in multi-agent RL, with classic cooperative methods such as COMA (Foerster et al., 2018) and QMIX (Rashid et al., 2020) proposing counterfactual or value-decomposition strategies for credit assignment among peer agents. In the LLM era, MAGRPO (Liu et al., 2026a), AT-GRPO (Zhao et al., 2025), M-GRPO (Hong et al., 2025), and Dr. MAS (Feng et al., 2026b) extend group-based RL to multi-agent systems through centralized advantages, trajectory alignment, or agent-wise normalization, while MATPO (Mo et al., 2025) and CoDA (Liu et al., 2026b) jointly train cooperating roles by broadcasting a shared advantage across decisions or splitting credit with fixed ratios. However, these methods treat agents as peers with independent action spaces or resort to static credit splits, leaving cross-role credit assignment in Planner–Executor architectures unresolved.

3 PRELIMINARIES

Group-based RL. Group-based Reinforcement Learning optimizes policies without an explicit value function by comparing a group of sampled outputs. Given an input x, the policy π_θ generates N candidate outputs $\{y_1, \ldots, y_N\}$, each receiving a scalar reward $r_i = R(x, y_i)$. Advantages are computed via within-group normalization, e.g., GRPO (Shao et al., 2024) standardizes rewards using the group mean and standard deviation. These advantages are then assigned uniformly to all tokens.

Multi-Turn Agentic RL. We model interactive tasks as multi-turn decision processes. Given an instruction Q, the agent observes state s_t and samples action $a_t \sim \pi_\theta(\cdot \mid s_t)$, producing a trajectory $\tau = (s_1, a_1, \ldots, s_T, a_T)$. The environment provides only a sparse outcome reward $R(\tau)$. Unlike single-output settings, the uniform advantage assignment in group-based RL cannot distinguish the contribution of individual steps, motivating the need for finer-grained credit assignment.

Problem Setup. We formalize Planner–Executor co-learning as a *Hierarchical Agentic RL* problem with two role policies: a planner π_{θ_P} generating a plan P, and an executor π_{θ_E} producing stepwise actions a_t conditioned on P. The environment returns only a sparse outcome reward R after T steps. The two policies are jointly optimized to maximize $J(\theta_P, \theta_E) = \mathbb{E}_{\tau \sim (\pi_{\theta_P}, \pi_{\theta_E})}[R]$.

4 METHODS

Motivation. Existing methods for training agents under Planner–Executor architecture either freeze one module while optimizing the other, or broadcast the same episodic advantage to all decisions uniformly. Both approaches blur the line between planning and execution, so when a trajectory fails, there is no way to tell whether the plan or the execution is to blame. We observe that the value of

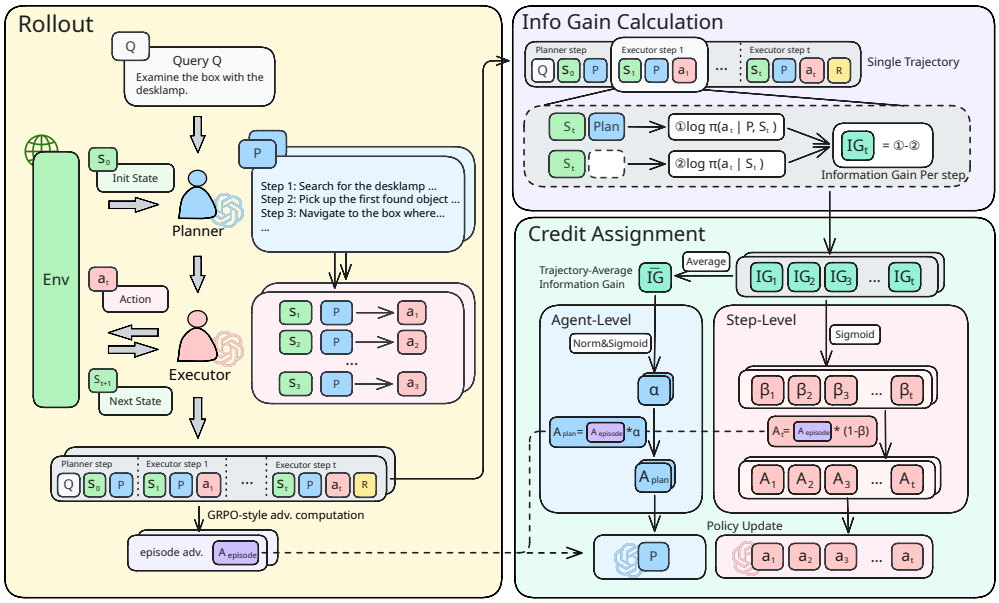


Figure 2: Overview of the Harlo framework. The **Rollout Phase** (left) collects plan-then-execute trajectories: the planner generates plan P from task Q, and the executor produces actions a_t conditioned on P; the outcome reward yields the episode-level advantage A_{episode} via group-relative normalization. The **Info Gain Calculation** phase (upper right) re-evaluates each executed action with and without the plan in context, producing step-level information gains $\{\text{IG}_1, \ldots, \text{IG}_T\}$. The **Credit Assignment** phase (lower right) converts these signals into two-level coefficients: the agent-level module maps the averaged gain $\bar{G}$ to the planner coefficient α, while the step-level module maps each IG_t to the executor coefficient γ_t. These coefficients modulate step-level advantages, which are combined with A_{episode} to produce the final advantages for updating both policies.

a plan should be judged by its effect on the executor's likelihood of producing correct actions: a plan that raises the probability of the correct action sequence deserves credit, whereas a plan that suppresses it should be penalized. This motivates a two-level credit assignment mechanism that uses information gain—the change in the executor's action likelihood induced by the plan—to allocate credit to the planner and executor steps based on whether the plan guides or misleads execution.

Overview. As illustrated in Figure 2, we propose Harlo, a hierarchical reinforcement learning framework with information-gain-guided credit assignment for Planner–Executor architectures. Harlo introduces a three-phase pipeline that first collects plan-then-execute trajectories, then computes counterfactual information gains, and finally performs two-level credit assignment at both the agent level and the step level. By leveraging information gain as an annotation-free proxy for planning influence, Harlo dynamically disentangles planner and executor contributions from outcome rewards.

4.1 Trajectory Collection and Episode-Level Advantage Computation

Given a task instruction Q and an initial observation o_0, we define the initial state as $s_0 = (Q, o_0)$. The planner policy π_{θ_P} generates a plan based on this initial state:

$$P^i \sim \pi_{\theta_P}(\cdot \mid s_0), \quad i = 1, \ldots, N. \tag{1}$$

For each plan P^i, the executor policy π_{θ_E} generates a complete trajectory by interacting with the environment for T^i steps. At each step $t \in \{1, \ldots, T^i\}$, the executor samples a low-level action conditioned on the plan and the current state:

$$a_t^i \sim \pi_{\theta_E}(\cdot \mid s_t^i, P^i), \tag{2}$$

where $s_t^i = (Q, o_t^i, h_{t-1}^i)$ encapsulates the observation o_t^i and execution history h_{t-1}^i. The environment returns a sparse outcome reward R^i upon episode termination, with zero intermediate rewards.

Each complete trajectory is recorded as:

$$\tau^i = (Q, s_0^i, P^i, s_1^i, a_1^i, \ldots, s_{T^i}^i, a_{T^i}^i, R^i). \tag{3}$$

To enable policy optimization from this sparse feedback, we assign the outcome reward to the planning decision and each executor step within the same trajectory:

$$r_{\text{plan}}^i = R^i, \qquad r_{\text{exec},t}^i = R^i, \quad t = 1, \ldots, T^i. \tag{4}$$

Following GRPO (Shao et al., 2024), we compute an episode-level advantage A_{episode}^i by normalizing the total return R^i within the group of N trajectories generated from the same task Q:

$$A_{\text{episode}}^i = \frac{R^i - \mu_R}{\sigma_R + \epsilon}, \tag{5}$$

where μ_R and σ_R denote the mean and standard deviation of the group rewards $\{R^1, \ldots, R^N\}$, respectively, and ϵ is a small constant for numerical stability. For notational brevity, we omit the trajectory index i in subsequent derivations.

4.2 Counterfactual Information Gain Computation

While the episode-level advantage provides effective group-relative comparison, it cannot distinguish the contributions of the planner from those of the executor. To address this, we quantify the plan's behavioral influence through *counterfactual information gain*—the change in the executor's action likelihood induced by the presence of the plan.

For each executor step $t \in \{1, \ldots, T\}$, we compute the information gain of the plan as the log-ratio of the executor's action probability under the planning context versus a counterfactual context where the plan is removed:

$$\text{IG}_t = \log \pi_{\theta_E}(a_t \mid s_t, P) - \log \pi_{\theta_E}(a_t \mid s_t, \varnothing), \tag{6}$$

where $\varnothing$ denotes the omission of the plan P from the executor's context, implemented by replacing the plan with a placeholder token "[No plan]". In practice, since each action a_t is a sequence of L_t tokens, we implement $\log \pi_{\theta_E}(a_t \mid \cdot)$ as the per-token average log-likelihood:

$$\log \pi_{\theta_E}(a_t \mid s_t, \cdot) := \frac{1}{L_t} \sum_{j=1}^{L_t} \log \pi_{\theta_E}(a_{t,j} \mid s_t, a_{t,<j}, \cdot), \tag{7}$$

where $a_{t,j}$ denotes the j-th token of action a_t. This token-level averaging ensures comparable IG_t across actions of varying lengths. The counterfactual computation requires no extra environment interaction—only a forward pass without the plan in context. Intuitively, positive IG_t indicates plan-guided actions, while negative IG_t suggests plan-misled decisions, providing an annotation-free measure of planning influence per step.

4.3 Two-Level Credit Assignment

The information gain sequence $\{\text{IG}_1, \ldots, \text{IG}_T\}$ provides step-level measures of planning influence. We perform credit assignment at two levels: the *agent level* for the planner, which assesses overall plan quality, and the *step level* for the executor, which distinguishes the contribution of each action.

4.3.1 Agent-Level Credit Assignment

We aggregate the information gains across all executor steps into a trajectory-average:

$$\bar{G}(\tau) = \frac{1}{T} \sum_{t=1}^{T} \text{IG}_t. \tag{8}$$

Since $\bar{G}(\tau)$ can vary significantly across tasks and model scales, we standardize it within the current training batch:

$$\tilde{G}(\tau) = \frac{\bar{G}(\tau) - \mu_G}{\sigma_G + \epsilon}, \tag{9}$$

where μ_G and σ_G are the mean and standard deviation of $\bar{G}$ over the batch. The planner contribution coefficient is then defined via a sigmoid function:

$$\alpha(\tau) = \frac{1}{1 + \exp(-\tilde{G}(\tau))}. \tag{10}$$

This coefficient serves as an agent-level credit scalar for the planner. Sigmoid maps standardized information gain to a belief probability of positive planning influence, serving as a continuous confidence measure for credit assignment.

The interpretation depends on the trajectory outcome: for a successful trajectory, a large positive information gain indicates that the plan effectively guided the executor toward correct actions and thus deserves amplified credit; a small or negative information gain suggests that the plan contributed little or even misled the executor, and its credit should be discounted.

Conversely, for a failed trajectory, a large positive information gain implies that the plan actively steered the executor toward incorrect actions and should therefore receive a larger penalty; a small or negative information gain indicates that the plan had little influence on the failure, warranting a milder penalty. The planner's advantage is modulated by this coefficient:

$$A_{\text{plan}} = A_{\text{episode}} \cdot \alpha(\tau), \tag{11}$$

This modulation ensures that agent-level credit is allocated to the planner proportionally to its demonstrable influence on execution, addressing the challenge of distinguishing planning failures from execution failures.

4.3.2 Step-Level Credit Assignment

We perform step-level credit assignment to quantify how much the plan influences each executor action. For each executor step t, we transform the local information gain using a sigmoid function:

$$\beta_t = \frac{1}{1 + \exp(-\text{IG}_t)}. \tag{12}$$

We then define the executor contribution coefficient:

$$\gamma_t = 1 - \beta_t. \tag{13}$$

This coefficient captures the step-level trade-off. A higher information gain means the plan strongly determines the action. Therefore, the executor's own contribution should be reduced. A lower information gain means the executor acts more autonomously. Thus, the executor deserves a larger share of credit. We modulate the executor's step-level advantage using this coefficient:

$$A_{\text{exec},t} = A_{\text{episode}} \cdot \gamma_t, \tag{14}$$

where A_{episode} is uniformly broadcast to every time step. This modulation enables fine-grained credit allocation at the step level. As a result, the executor policy receives a clear, step-wise optimization signal. This signal reflects its true contribution to the final trajectory outcome.

4.4 Overall Policy Optimization

With the two-level credit signals defined, we integrate them into a unified policy update. The planner receives a single trajectory-level advantage, while the executor obtains step-wise advantages that reflect the contribution of each action:

$$A_{\text{plan}}^{\text{final}} = A_{\text{episode}} + w \cdot A_{\text{plan}}, \tag{15}$$

$$A_{\text{exec},t}^{\text{final}} = A_{\text{episode}} + w \cdot A_{\text{exec},t}, \quad t = 1, \ldots, T. \tag{16}$$

Here, w is a hyperparameter controlling the influence of the credit-modulated terms, which is set to 1 by default. The final advantages are then used to optimize both policies with a group-based policy objective. The planner loss is:

$$\mathcal{L}(\theta_P) = -\,\mathbb{E}_\tau \left[g(\rho_P(\theta_P), A_{\text{plan}}^{\text{final}}) \right] + \beta\, \mathbb{D}_{\text{KL}} \left(\pi_{\theta_P} \,\|\, \pi_{\text{ref}} \right), \tag{17}$$

and the executor loss is:

$$\mathcal{L}(\theta_E) = -\,\mathbb{E}_\tau \left[\frac{1}{T} \sum_{t=1}^{T} g(\rho_E^{(t)}(\theta_E), A_{\text{exec},t}^{\text{final}}) \right] + \beta\, \mathbb{D}_{\text{KL}} \left(\pi_{\theta_E} \,\|\, \pi_{\text{ref}} \right), \tag{18}$$

where ρ_P and $\rho_E^{(t)}$ are the importance ratios for the planner and each executor step. The clipping operation constrains large policy changes, while the KL term regularizes updates toward the reference model. This design enables stable co-evolution of planning and execution during joint training.

5 EXPERIMENTS

In this section, we conduct comprehensive experiments to evaluate Harlo on widely used interactive benchmarks, focusing on the following research questions (RQs).

- **RQ1:** How does Harlo perform compared to representative baselines on interactive reasoning benchmarks?
- **RQ2:** How do the agent-level and step-level credit assignment components influence the learning dynamics and final performance of Harlo?
- **RQ3:** How does the information gain signal evolve during training under Harlo, and how does its trend compare with that of baseline methods?
- **RQ4:** What is the additional computational cost introduced by Harlo's information-gain computation compared to standard GRPO and other methods?

5.1 EXPERIMENT SETUP

Benchmarks. We evaluate Harlo on three interactive environments: **ALFWorld** (Shridhar et al., 2020), a text-based embodied household environment; **ScienceWorld** (Wang et al., 2022), a virtual lab simulation requiring multi-step causal reasoning; and **WebShop** (Yao et al., 2022a), an open-ended web shopping environment. For ALFWorld and ScienceWorld, we adopt the L0–L2 splits following prior work (Zhang et al., 2025).

Baselines. We compare Harlo against four categories of methods. *Prompt-based / Non-RL flat methods:* ReAct (Yao et al., 2022b), SFT, and ETO (Song et al., 2024). *Prompt-based hierarchical methods:* ADaPT (Prasad et al., 2024). *RL-based flat methods:* GRPO (Shao et al., 2024) and GiGPO (Feng et al., 2026a). *RL-based hierarchical methods:* EAGLET (Si et al., 2026) and RLVMR (Zhang et al., 2025). We also report closed-source models (GPT-4o (Achiam et al., 2023), GPT-5-mini (Singh et al., 2025), GPT-5 (Singh et al., 2025), Gemini-2.5-Pro (Comanici et al., 2025), Gemini-2.5-Flash (Comanici et al., 2025)) as references.

Evaluation metrics. For ALFWorld and ScienceWorld, we report success rate (%) based on goal completion. For WebShop, we report both score and success rate (%). All results are averaged over three random seeds, with each evaluation conducted on 128 test episodes per split.

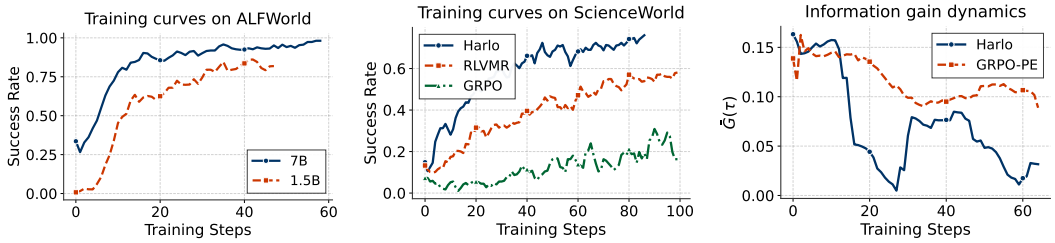
Implementation Details. We use Qwen-2.5-1.5B and 7B (Qwen et al., 2025) as base models, with planner and executor distinguished by different system prompts. All RL methods are warm-started from a shared SFT checkpoint. The SFT data consists of expert trajectories from AgentGym (Xi et al., 2025), which GPT-4o re-annotates into the Planner-Executor format. ALFWorld and WebShop are trained for 60 iterations, while ScienceWorld is trained for 100 iterations, with early stopping based on validation performance. More details are provided in the Appendix A.

5.2 OVERALL PERFORMANCE COMPARISONS (RQ1)

Table 1 presents the comparison between Harlo and baselines across ALFWorld, ScienceWorld, and WebShop. Single-agent RL methods (GRPO, GiGPO) consistently outperform ReAct and SFT, validating the effectiveness of RL post-training. Hierarchical RL methods like RLVMR and EAGLET further push performance by leveraging planner–executor structures. However, these methods broadcast uniform advantages or rely on fixed credit splits, failing to distinguish planning quality from execution proficiency. In contrast, Harlo dynamically allocates credit based on actual influence.

Table 1: Main results on ALFWorld, ScienceWorld, and WebShop across three difficulty levels. Success rates (and score for WebShop) are reported, with best scores in bold.

Model	Type	Method	ALFWorld			ScienceWorld			WebShop	
			L0	L1	L2	L0	L1	L2	score	succ.
GPT-4o	Flat	ReAct	52.3	48.4	38.3	32.0	32.8	32.8	12.7	6.2
GPT-5-mini	Flat	ReAct	57.0	54.7	46.1	39.8	31.3	31.4	14.2	7.0
GPT-5	Flat	ReAct	70.0	60.2	63.3	53.9	33.6	33.6	34.9	29.7
Gemini-2.5-Pro	Flat	ReAct	56.3	55.0	55.5	32.8	27.3	30.5	42.9	32.0
Gemini-2.5-Flash	Hierarchical	ADaPT	62.5	56.8	54.2	35.8	31.4	31.5	41.6	31.6
Qwen-2.5-1.5B	Flat	ReAct	11.3	13.7	10.2	1.2	0.8	0.8	40.1	11.3
	Flat	SFT	43.0	38.7	17.6	20.3	18.0	12.5	65.8	39.1
	Flat	ETO	64.1	66.4	25.8	39.1	22.7	15.6	68.4	42.5
	Flat	GRPO	76.6	71.1	29.7	21.1	13.7	10.9	75.8	56.8
	Flat	GiGPO	86.7	83.2	48.0	25.8	15.2	4.7	83.1	65.0
	Hierarchical	EAGLET	88.9	**88.6**	58.5	56.3	43.8	33.2	**87.7**	**73.8**
	Hierarchical	RLVMR	89.1	87.9	56.3	46.9	34.4	26.5	86.7	71.1
	Hierarchical	**Harlo**	**90.6**	88.3	**67.1**	**62.5**	**53.1**	**42.2**	87.0	73.3
		(vs. GRPO)	+18%	+24%	+126%	+196%	+288%	+287%	+15%	+29%
Qwen-2.5-7B	Flat	ReAct	23.1	28.5	27.0	7.8	11.3	6.3	46.2	19.5
	Flat	SFT	63.3	57.0	37.5	36.7	32.0	23.4	62.2	38.3
	Flat	ETO	70.3	74.2	51.6	62.5	40.6	28.1	69.7	44.8
	Flat	GRPO	79.3	77.3	52.3	49.1	30.1	26.6	79.3	66.1
	Flat	GiGPO	89.5	90.2	67.2	53.4	35.2	25.8	84.4	72.8
	Hierarchical	EAGLET	92.0	93.3	82.8	62.1	51.4	39.5	88.2	79.6
	Hierarchical	RLVMR	91.4	91.8	83.6	67.2	43.0	32.2	87.3	74.2
	Hierarchical	**Harlo**	**96.9**	**98.2**	**85.1**	**68.0**	**62.5**	**49.2**	**89.9**	**81.3**
		(vs. GRPO)	+22%	+27%	+63%	+38%	+108%	+85%	+13%	+23%



(a) (b) (c)

Figure 3: Training dynamics and information gain analysis. (a) Training curves on ALFWorld. (b) Training curves on ScienceWorld. (c) Information gain dynamics during training. More training curves are provided in the Appendix B.

This design yields substantial gains across all benchmarks and model scales. Notably, improvements are most pronounced on challenging long-horizon splits. Harlo even surpasses proprietary models like GPT-5 and Gemini-2.5-Pro on several splits. Figure 3(a)–(b) show the training dynamics on ALFWorld and ScienceWorld, where Harlo achieves faster convergence and higher performance. Overall, Harlo achieves competitive performance across diverse interactive reasoning benchmarks.

5.3 Ablation Studies (RQ2)

To examine how the agent-level and step-level credit assignment components influence Harlo's performance, we conduct ablation studies using the 1.5B backbone.

As shown in Table 2, removing step-level credit assignment (w/o step C.A.) causes notable performance drops, indicating that fine-grained step modulation is essential for distinguishing plan-shaped actions from executor-driven ones in long-horizon tasks. Removing both credit assignment levels

(w/o agent+step C.A.) reverts to GRPO-style uniform advantage, yielding even worse performance and confirming that agent-level modulation alone provides an insufficient foundation without step-level complement.

The gap between ablations widens with task difficulty, demonstrating that both components are critical in complex scenarios.

Table 2: Ablation study on ScienceWorld

Variant	L0	L1	L2
Harlo (full)	**62.5**	**53.1**	**42.2**
w/o step C.A.	60.9	52.6	37.5
w/o agent+step C.A.	58.5	52.3	35.9

5.4 Information Gain Dynamics (RQ3)

To investigate how Harlo influences the interaction between the planner and executor during training, we track the trajectory-average information gain $\bar{G}(\tau)$, which measures the average behavioral influence of the plan over the entire trajectory. We compare its evolution during training between Harlo and a GRPO variant that employs a Planner-Executor architecture but performs uniform credit assignment (denoted as GRPO-PE), using the ScienceWorld benchmark with the 1.5B backbone.

As shown in Figure 3(c), Harlo exhibits a distinctive non-monotonic evolution pattern. The information gain first decreases rapidly during early training, indicating that the executor gradually reduces its dependence on explicit planning guidance. After reaching a lower level, the information gain increases again, suggesting that the planner continues to improve and provides increasingly effective guidance to the executor. This adaptive fluctuation reflects the evolving interaction pattern between the two modules, where the executor gradually becomes less dependent on explicit guidance while the planner continues to provide more informative plans. In contrast, GRPO-PE maintains consistently higher information gain without a clear adaptation trend, indicating less effective coordination between planning and execution. These results demonstrate that Harlo facilitates dynamic adjustment of the planner–executor interaction during training, enabling the two modules to achieve a more effective division of responsibilities.

5.5 Computational Cost Analysis (RQ4)

To assess the practical efficiency of Harlo, we benchmark its per-iteration runtime against GRPO. As detailed in Section 4.2, the information gain computation requires no extra environment interaction—only forward passes through the executor policy with the plan omitted.

As shown in Figure 4, which compares the per-iteration time breakdown of GRPO, Harlo, and EAGLET, the rollout phase remains the dominant bottleneck, while the newly introduced components introduce only modest overhead (about 20% of total time). In contrast to EAGLET (Si et al., 2026), which doubles rollout costs by regenerating complete trajectories without the plan, our approach computes information gain from log-probabilities of executed actions. This design offers a trade-off between performance and efficiency, making Harlo suitable for real-world deployment.

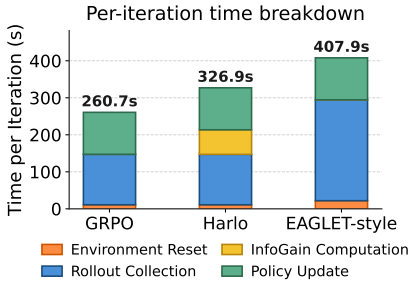


Figure 4: Per-iteration time breakdown.

6 Conclusion

We propose Harlo, a hierarchical agentic RL framework that addresses the credit assignment problem in Planner–Executor architectures through information-gain-guided two-level credit assignment. Experiments on ALFWorld, ScienceWorld, and WebShop demonstrate that Harlo consistently outperforms strong baselines by dynamically disentangling planning and execution contributions. A limitation of the current work is its focus on the single-planner-executor setting; extending Harlo to more general multi-agent hierarchies remains an important direction for future work.

AI USE STATEMENT

We used generative AI tools for the following tasks: assisting with the implementation of our method and the interpretation of our experimental results; generating synthetic data, namely the cold-start supervised fine-tuning data described in Appendix A.2, which was produced by prompting GPT-4o at temperature 0 to augment existing expert trajectories with an explicit planning step (299 trajectories for ALFWorld, 297 for ScienceWorld, and 300 for WebShop); identifying relevant related work and formatting references; and polishing the manuscript to improve its readability. All AI-assisted outputs, including the generated planning annotations, were reviewed and verified by the authors against the source trajectories and the original cited sources, and we take full responsibility for the final content of this work.

ETHICS STATEMENT

Our work trains and evaluates LLM-based agents in three publicly available simulated text environments accessed through their original codebases (Appendix D). No human subjects or personally identifiable information were involved, and no IRB approval was required. All environments are sandboxes: WebShop, in particular, operates on a synthetic store from a public product catalogue and performs no real transactions, so a policy trained in this work cannot execute, or be rewarded for, an actual purchase. We note one potential source of dual use: a policy that learns to plan and act over long horizons in a web environment could in principle be repurposed to automate interfaces not intended for automation. We consider this a limited step in that direction rather than an immediate hazard, since the demonstrated behaviours are confined to simulated benchmarks and the policies are 1.5B and 7B parameter models. We do not foresee any other ethical concerns arising from this work.

REPRODUCIBILITY STATEMENT

We describe our efforts to ensure reproducibility below. The base models, optimizer settings, sampling configuration, per-episode step budgets, batch sizes, hardware configuration, and the exact random seeds behind the three independent runs are reported in Appendix A; the corresponding evaluation protocol is stated in Section 5.1. The construction of the cold-start supervised fine-tuning data, including the prompt template used to produce the planning annotations and the resulting dataset sizes, is documented in Appendix A.2. The L0–L2 split definitions, success criteria, and reward binarization rules for each environment are given in Appendix D. The complete prompts used by the planner and the executor are listed in Appendix E. Descriptions of all baseline methods are provided in Appendix C. Worked examples of the information-gain computation, including per-step gains, the resulting credit coefficients, and the batch statistics used for standardization, appear in Appendix F, and additional training curves are given in Appendix B. The source code is provided in the anonymized supplementary material.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026a.

Lang Feng, Longtao Zheng, Shuo He, Fuxiang Zhang, and Bo An. Dr. mas: Stable reinforcement learning for multi-agent llm systems. *arXiv preprint arXiv:2602.08847*, 2026b.

Yuming Feng, Chuye Hong, Yaru Niu, Shiqi Liu, Yuxiang Yang, and Ding Zhao. Learning multi-agent loco-manipulation for long-horizon quadrupedal pushing. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14441–14448. IEEE, 2025.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ali Essam Ghareeb, Benjamin Chang, Ludovico Mitchener, Angela Yiu, Caralyn J Szostkiewicz, Dmytro Shved, Gavin J Gyimesi, Jon M Laurent, Samantha M Wright, Muhammed T Razzak, et al. A multi-agent system for automating scientific discovery. *Nature*, pp. 1–3, 2026.

Haoyang Hong, Jiajun Yin, Yuan Wang, Jingnan Liu, Zhe Chen, Ailing Yu, Ji Li, Zhiling Ye, Hansong Xiao, Yefei Chen, et al. Multi-agent deep research: Training multi-agent systems with m-grpo. *arXiv preprint arXiv:2511.13288*, 2025.

Fanqi Kong, Jiayi Zhang, Mingyi Deng, Chenglin Wu, Yuyu Luo, and Bang Liu. Infopo: Information-driven policy optimization for user-centric agents. *arXiv preprint arXiv:2603.00656*, 2026.

Zhongyi Li, Wan Tian, Jinju Chen, Huiming Zhang, Yang Liu, Yikun Ban, and Fuzhen Zhuang. Counterfactual credit policy optimization for multi-agent collaboration. *arXiv preprint arXiv:2603.21563*, 2026.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. Llm collaboration with multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32150–32158, 2026a.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Junge Zhang, and Jianbin Jiao. Agentic reinforcement learning with implicit step rewards. *arXiv preprint arXiv:2509.19199*, 2025.

Xuanzhang Liu, Jianglun Feng, Zhuoran Zhuang, Junzhe Zhao, Maofei Que, Jieting Li, Dianlei Wang, Hao Tong, Ye Chen, and Pan Li. Coda: A context-decoupled hierarchical agent with reinforcement learning. In *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining*, pp. 436–446, 2026b.

Keer Lu, Chong Chen, Xili Wang, Bin Cui, Yunhuai Liu, and Wentao Zhang. Pilotrl: Training language model agents via global planning-guided progressive reinforcement learning. *arXiv preprint arXiv:2508.00344*, 2025.

Zhanfeng Mo, Xingxuan Li, Yuntao Chen, and Lidong Bing. Multi-agent tool-integrated policy optimization. *arXiv preprint arXiv:2510.04678*, 2025.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

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

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

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

Shuzheng Si, Haozhe Zhao, Kangyang Luo, Gang Chen, Fanchao Qi, Minjia Zhang, Baobao Chang, and Maosong Sun. A goal without a plan is just a wish: Efficient and effective global planner training for long-horizon agent task. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13086–13113, 2026.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245, 2023.

Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, and Zhenzhe Ying. Information gain-based policy optimization: A simple and effective approach for multi-turn search agents. In *The Fourteenth International Conference on Learning Representations*, 2026.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, et al. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

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

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. *arXiv preprint arXiv:2507.22844*, 2025.

Yuan Zhao, Hualei Zhu, Tingyu Jiang, Shen Li, Xiaohang Xu, and Hao Henry Wang. Co-epg: A framework for co-evolution of planning and grounding in autonomous gui agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36582–36590, 2026.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-mas: Multi-agent reinforcement learning for collaborative llms. *arXiv preprint arXiv:2510.11062*, 2025.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.

A Experiment Details

A.1 Experiment Setup

We use Qwen-2.5-1.5B and Qwen-2.5-7B as base models. The planner and the executor share a single set of parameters. The first step is a planning step that outputs the entire plan within <planning> tags, and every subsequent step is an execution step that outputs one action within <action> tags, with the two roles conditioned by the role-specific prompts. Both roles are initialized by cold-start supervised fine-tuning before reinforcement learning.

Training is implemented on top of the verl framework with FSDP, bf16 precision, and full-parameter fine-tuning, using vLLM as the rollout engine. The policy is optimized with AdamW at a constant learning rate of 1×10^{-6} without warmup, and gradients are clipped to norm 1.0. Experiments are conducted on 2 NVIDIA H20 GPUs for the 1.5B model and 4 NVIDIA H20 GPUs for the 7B model.

For each task, we sample $N = 8$ plan-then-execute trajectories with temperature 1.0 and top-p 1.0. The maximum prompt length is 6,000 tokens and the maximum generation length is 256 tokens per step. The executor's context contains the most recent two observation–action pairs as interaction history. The maximum number of environment interaction steps per episode is 30 for ALFWorld, 30 for ScienceWorld, and 10 for WebShop. Each training iteration collects 16 tasks, i.e., 128 trajectories, with a PPO mini-batch size of 256 and a per-GPU micro-batch size of 16.

The policy is updated with a clipped surrogate objective (clip ratio 0.2), KL regularization toward the reference model with coefficient 0.01 (low-variance KL estimator), and an entropy bonus of 0.001. The credit weight is $w = 1.0$, and the numerical stability constant used in all standardizations is 10^{-6}. ALFWorld and WebShop are trained for 60 iterations and ScienceWorld for 100 iterations, with checkpoints saved every 10 iterations and the reported model selected by validation performance.

Evaluation uses 128 episodes per split with sampled decoding at temperature 0.4. All reported results are averaged over three independent runs with random seeds 0, 24, and 42, where each run repeats the full training and evaluation pipeline under its seed; in each run, evaluation environments are seeded with the training seed offset by 1,000.

A.2 Cold-Start Supervised Fine-Tuning

Before reinforcement learning, both roles are fine-tuned on expert trajectories collected from AgentGym (and the corresponding environment-specific expert data), re-annotated into the plan-then-execute format. The underlying trajectories are expert demonstrations rather than GPT-4o rollouts; GPT-4o is used only as an offline annotator at temperature 0, which is prompted with the complete trajectory and a tagging template and returns step-wise annotations whose first step carries the planning content. This yields 299 trajectories (4,250 steps) for ALFWorld, 297 trajectories (5,083 steps) for ScienceWorld, and 300 trajectories (1,813 steps) for WebShop. Each step's prompt–response pair is used as one training example. The SFT stage runs for 3 epochs with AdamW (learning rate 1×10^{-5}, betas $(0.9, 0.95)$, weight decay 0.01), a cosine schedule with warmup ratio 0.1, gradient clipping 1.0, and maximum sequence lengths of 2,048–4,192 tokens depending on the environment.

B Training Dynamics

Figure 5 presents that across all three environments, the 7B model consistently achieves faster early convergence and higher final success rates, confirming that increased model capacity benefits hierarchical agentic RL. The gap is most pronounced on ALFWorld, where the 7B model saturates near 100% after roughly 40 training steps while the 1.5B model converges more slowly to approximately 85%. On WebShop, the 7B model maintains a stable lead of about 10 percentage points throughout training. In contrast, ScienceWorld shows the smallest scale sensitivity: although the 7B model rises more quickly in early iterations, the 1.5B model eventually catches up to a comparable final performance, suggesting that scientific procedural reasoning tasks are less dependent on raw model capacity once sufficient training is provided. These dynamics are consistent with the final evaluation results reported in the main text and indicate that Harlo trains stably across model scales, with larger

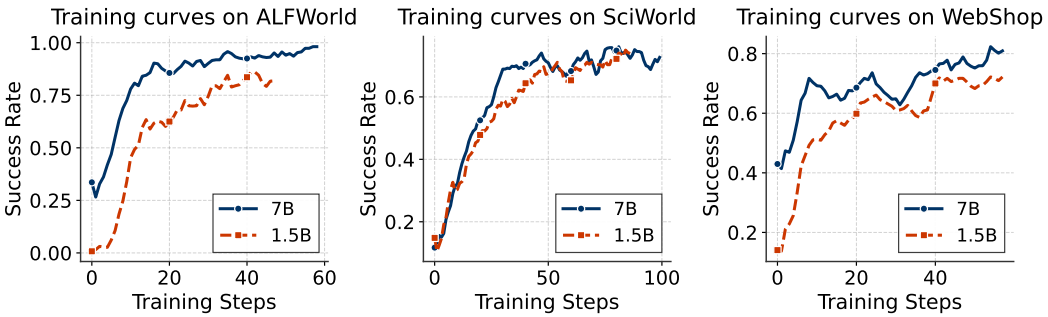


Figure 5: Training success rate curves of Harlo with 1.5B and 7B models on ALFWorld, ScienceWorld and WebShop.

backbones offering disproportionate gains on complex, long-horizon household and web-interaction tasks.

C Baseline Methods

ReAct. ReAct prompts a single LLM to interleave free-form reasoning traces with environment actions in one generation loop, so that each action is preceded by an explicit thought. It involves no parameter updates and serves as the standard prompting baseline for interactive tasks; the closed-source models in the main table (GPT-4o, GPT-5-mini, GPT-5, Gemini-2.5-Pro) are evaluated under this paradigm as reference points.

SFT. Supervised fine-tuning performs behavior cloning on expert trajectories: the model is trained with a standard cross-entropy objective to imitate the recorded action at every step, without any environment interaction or reward signal during training.

ETO. ETO (Exploration-based Trajectory Optimization) learns from failures. The agent explores the environment, and its failed trajectories are paired with expert or successful ones to form trajectory-level preferences, which are then optimized with a DPO-style contrastive objective over iterative explore-then-update rounds.

GRPO. GRPO is a critic-free group-based RL algorithm. For each task it samples a group of rollouts, standardizes their outcome rewards by the group mean and standard deviation to obtain a single trajectory-level advantage, and broadcasts this advantage uniformly to all tokens of the rollout, so that every decision in a trajectory receives the same learning signal.

GiGPO. GiGPO (Group-in-Group Policy Optimization) refines GRPO with a second, finer level of grouping: actions taken from the same anchor state across different rollouts of the same task are grouped together, yielding step-level relative advantages in addition to the episode-level ones, without extra rollouts or a learned critic.

ADaPT. ADaPT (As-Needed Decomposition and Planning) is a prompt-based hierarchical method. An executor LLM first attempts a (sub-)task directly, and only upon failure does a planner LLM recursively decompose it into simpler sub-tasks. Decomposition therefore happens as needed at inference time, with no training involved.

EAGLET. EAGLET trains a plug-and-play global planner while keeping the executor frozen. Candidate plans are scored by contrasting the frozen executor's rollouts with and without the plan in context, and the planner is optimized toward plans that improve execution. Because the executor is never updated, the two roles do not co-evolve during training.

RLVMR. RLVMR augments the sparse outcome reward with dense, verifiable meta-reasoning rewards. The agent explicitly tags each step with a cognitive phase (e.g., planning, exploration,

reflection), and rule-based checks grant rewards for verifiably useful reasoning behaviors—such as avoiding repeated exploration—so that sound reasoning processes are reinforced alongside task success rather than outcomes alone.

D DATASET DETAILS

We evaluate Harlo on three interactive environments, each accessed through its original codebase. For ALFWorld and ScienceWorld we adopt the L0–L2 generalization splits following RLVMR; WebShop has no difficulty levels and uses a single train/test division. Each environment is trained with a separate policy.

D.1 ALFWORLD

ALFWorld is a text-based embodied household environment that aligns the TextWorld engine with the ALFRED benchmark. Given a natural-language goal such as "put a clean mug on the desk", the agent interacts through text commands (e.g., `go to`, `take`, `open`, `clean`, `heat`) to navigate among receptacles and manipulate objects. Tasks fall into six categories—Pick & Place, Examine in Light, Clean & Place, Heat & Place, Cool & Place, and Pick Two & Place—and require completing several sub-goals in the correct order over long horizons, making the environment a canonical testbed for hierarchical decision making, where a high-level plan must be faithfully grounded into low-level admissible actions.

We train on the full ALFWorld training set (TextWorld games, version 2.1.1) with a maximum of 30 interaction steps per episode; the environment returns an outcome reward of 10 upon task success and 0 otherwise. The three evaluation levels are constructed as follows. L0 evaluates on in-distribution episodes drawn from the same configuration as training. L1 evaluates on unseen environment instances (the `valid_unseen` split) under the same task categories. L2 adopts a task-level out-of-distribution regime in which training covers only the Pick & Place, Examine in Light, Clean & Place, and Heat & Place categories, and evaluation consists solely of the two held-out categories, Cool & Place and Pick Two & Place, so the L2 policy is trained separately from the L0/L1 policy. Each split is evaluated on 128 episodes.

D.2 SCIENCEWORLD

ScienceWorld is an interactive text environment that simulates a multi-room science laboratory, covering 30 task types across 10 scientific topics such as thermodynamics, electricity, states of matter, classification, and biology. Completing a task requires executing a multi-step scientific procedure—for example, locating instruments, preparing materials, and measuring the melting point of a substance—under strict procedural dependencies, with horizons substantially longer than those in ALFWorld. The environment tracks progress through sub-goal completion, which makes it particularly suitable for assessing whether a plan decomposes a task into the correct sequence of stages.

We use the environment with the "easy" simplification preset and a maximum of 30 interaction steps per episode. An episode is counted as successful only if the task is completed with a full score of 100, in which case the outcome reward is 10 and otherwise 0; the environment's raw sub-goal score is not used as a training signal. The splits are constructed as follows. L0 contains 3,322 training and 1,661 test (task, variation) pairs over 29 task types, where training and test share the same task–variation pool. L1 contains 3,322 training and 1,684 test pairs over the same 29 task types, with variations unseen during training. L2 contains 2,242 training pairs over 19 task types and 549 test pairs over 10 task types unseen during training. Each split is evaluated on 128 episodes.

D.3 WEBSHOP

WebShop is a simulated online-shopping environment built from over one million real-world products. Given an instruction specifying target attributes, options, and a price ceiling, the agent issues search queries and click actions to locate and purchase a matching item. The final reward is computed from the match between the purchased product and the instruction in terms of attributes, options, and price, and a purchase satisfying all requirements is counted as a success.

We use the text environment with a maximum of 10 interaction steps per episode. Training and test instructions are disjoint by construction: training episodes draw instruction seeds below 2^{16} and test episodes draw seeds at or above it. When an observation exceeds 13,000 characters, the executor context falls back to a history-free template. For training, the environment's continuous matching score is binarized: an episode receives a outcome reward of 10 only if the purchased product perfectly matches the instruction (original environment score equal to 1), and 0 otherwise. At evaluation time we report both the success rate and the average of the original continuous task score. Evaluation uses 128 episodes.

E PROMPTS

This section lists the complete prompts used by Harlo. Fields in curly braces (e.g., {task_description}) are filled at run time; OBJ and LOC in the ScienceWorld action definitions are placeholders for concrete objects and locations. The planner prompt is used only at the first step of each rollout and the executor prompt at every subsequent step.

E.1 PLANNER PROMPTS

The planner receives the task instruction and the initial observation, and must output the entire high-level plan within <planning> tags, without any action tags.

ALFWorld

> You are an expert PLANNER operating in the ALFRED Embodied Environment. Your current task is: {task_description}
> Your current observation is: {current_observation} Your admissible actions of the current situation are: [{admissible_actions}].
> First, plan the entire task by breaking it down into high-level steps within <planning> </planning> tags. Focus on outlining the full sequence required to complete the overall task, not just the immediate next action. It is necessary to list all the points separately. eg, step 1: xxx, step 2: xxx, step 3: xxx, etc.
> You MUST output your plan within <planning> </planning> tags only. Do not output any action tags.

ScienceWorld

> You are an expert PLANNER operating in the ScienceWorld environment, which is a text-based virtual environment centered around accomplishing tasks from the elementary science curriculum. Your current task is: {task_description}
> Your current observation is: {current_observation}
> As a PLANNER, your role is to analyze the task and create a comprehensive plan. You do not execute actions yourself - you only provide the plan for the EXECUTOR to follow.
> Here are the actions you may take:
> { "action": "open OBJ", "description": "open a container" },
> { "action": "close OBJ", "description": "close a container" },
> { "action": "activate OBJ", "description": "activate a device" },
> { "action": "deactivate OBJ", "description": "deactivate a device" },
> { "action": "connect OBJ to OBJ", "description": "connect electrical components" },
> { "action": "disconnect OBJ", "description": "disconnect electrical components" },
> { "action": "use OBJ [on OBJ]", "description": "use a device/item" },
> { "action": "look around", "description": "describe the current room" },
> { "action": "look at OBJ", "description": "describe an object in detail" },
> { "action": "look in OBJ", "description": "describe a container's contents" },
> { "action": "read OBJ", "description": "read a note or book" },
> { "action": "move OBJ to OBJ", "description": "move an object to a container" },
> { "action": "pick up OBJ", "description": "move an object to the inventory" },
> { "action": "put down OBJ", "description": "drop an inventory item" },

{ ”action”: ”pour OBJ into OBJ”, ”description”: ”pour a liquid into a container” },
{ ”action”: ”dunk OBJ into OBJ”, ”description”: ”dunk a container into a liquid” },
{ ”action”: ”mix OBJ”, ”description”: ”chemically mix a container” },
{ ”action”: ”go to LOC”, ”description”: ”move to a new location” },
{ ”action”: ”eat OBJ”, ”description”: ”eat a food” },
{ ”action”: ”flush OBJ”, ”description”: ”flush a toilet” },
{ ”action”: ”focus on OBJ”, ”description”: ”signal intent on a task object” },
{ ”action”: ”wait”, ”description”: ”take no action for 10 iterations” },
{ ”action”: ”wait1”, ”description”: ”take no action for 1 iteration” },
{ ”action”: ”task”, ”description”: ”describe current task” },
{ ”action”: ”inventory”, ”description”: ”list your inventory” }
Current available actions: {available_actions}
First, plan the entire task by breaking it down into high-level steps within <planning> </planning> tags. Focus on outlining the full sequence required to complete the overall task, not just the immediate next action. It is necessary to list all the points separately. eg, step 1: xxx, step 2: xxx, step 3: xxx, etc.
You MUST output your plan within <planning> </planning> tags only. Do not output any action tags.

WebShop

You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your current task is: {task_description}
Your current observation is: {current_observation}
As a PLANNER, your role is to analyze the task and create a comprehensive plan. You do not execute actions yourself - you only provide the plan for the EXECUTOR to follow.
Your admissible actions of the current situation are: [{available_actions}].
First, plan the entire task by breaking it down into high-level steps within <planning> </planning> tags. Focus on outlining the full sequence required to complete the overall task, not just the immediate next action. It is necessary to list all the points separately. <planning> step 1: xxx, step 2: xxx, step 3: xxx, etc. </planning>
You MUST output your plan within <planning> </planning> tags only. Do not output any action tags.

E.2 Executor Prompts

At each execution step, the executor receives the task instruction, the most recent two observation–action pairs, the current observation, the admissible actions, and the plan, and must output exactly one action within <action> tags.

ALFWorld

You are an expert EXECUTOR operating in the ALFRED Embodied Environment. Your current task is: {task_description}
Prior to this step, you have already taken {step_count} step(s). Below are the most recent {history_length} observations and the corresponding actions you took: {action_history} You are now at step {current_step} and your current observation is: {current_observation} Your admissible actions of the current situation are: [{admissible_actions}].
Your previous overall plan is: {planning}. Please strictly adhere to your plan.
As an EXECUTOR, your role is to execute actions based on the plan provided by the PLANNER. You should NOT output any reasoning tags - simply select and execute the most appropriate action for the current step.
You MUST output your selected action within <action> </action> tags only.
Specify the next action following these guidelines: 1. Object and Receptacle References: Use specific identifiers: - [obj id] for objects (e.g., apple 1). - [recep id] for receptacles (e.g., countertop 1). 2. Action Validity: Follow the exact format below. Any deviation renders the

action invalid: Valid actions: go to [recep id], take [obj id] from [recep id], put [obj id] in/on [recep id], open/close [recep id], use [obj id], heat/cool/clean [obj id] with [recep id] <action> Choose the most appropriate action from the valid actions. </action>

ScienceWorld

You are an expert EXECUTOR operating in the ScienceWorld environment, which is a text-based virtual environment centered around accomplishing tasks from the elementary science curriculum. Your current task is: {task_description}
Prior to this step, you have already taken {step_count} step(s). Below are the most recent {history_length} observations and the corresponding actions you took: {action_history} You are now at step {current_step} and your current observation is: {current_observation} Here are the actions you may take:
{ "action": "open OBJ", "description": "open a container" },
{ "action": "close OBJ", "description": "close a container" },
{ "action": "activate OBJ", "description": "activate a device" },
{ "action": "deactivate OBJ", "description": "deactivate a device" },
{ "action": "connect OBJ to OBJ", "description": "connect electrical components" },
{ "action": "disconnect OBJ", "description": "disconnect electrical components" },
{ "action": "use OBJ [on OBJ]", "description": "use a device/item" },
{ "action": "look around", "description": "describe the current room" },
{ "action": "look at OBJ", "description": "describe an object in detail" },
{ "action": "look in OBJ", "description": "describe a container's contents" },
{ "action": "read OBJ", "description": "read a note or book" },
{ "action": "move OBJ to OBJ", "description": "move an object to a container" },
{ "action": "pick up OBJ", "description": "move an object to the inventory" },
{ "action": "put down OBJ", "description": "drop an inventory item" },
{ "action": "pour OBJ into OBJ", "description": "pour a liquid into a container" },
{ "action": "dunk OBJ into OBJ", "description": "dunk a container into a liquid" },
{ "action": "mix OBJ", "description": "chemically mix a container" },
{ "action": "go to LOC", "description": "move to a new location" },
{ "action": "eat OBJ", "description": "eat a food" },
{ "action": "flush OBJ", "description": "flush a toilet" },
{ "action": "focus on OBJ", "description": "signal intent on a task object" },
{ "action": "wait", "description": "take no action for 10 iterations" },
{ "action": "wait1", "description": "take no action for 1 iteration" },
{ "action": "task", "description": "describe current task" },
{ "action": "inventory", "description": "list your inventory" }
Your previous overall plan is: {planning}. Please strictly adhere to your plan.
As an EXECUTOR, your role is to execute actions based on the plan provided by the PLANNER. You should NOT output any reasoning tags - simply select and execute the most appropriate action for the current step.
You MUST output your selected action within <action> </action> tags only.

WebShop

You are an expert EXECUTOR agent operating in the WebShop e-commerce environment. Your task is to: {task_description}. Prior to this step, you have already taken {step_count} step(s). Below are the most recent {history_length} observations and the corresponding actions you took: {action_history} You are now at step {current_step} and your current observation is: {current_observation}. Your admissible actions of the current situation are: [{available_actions}].
Your previous overall plan is: {planning}. Please strictly adhere to your plan.
As an EXECUTOR, your role is to execute actions based on the plan provided by the PLANNER. You should NOT output any reasoning tags - simply select and execute the most appropriate action for the current step.

You MUST output your selected action within <action> </action> tags only.

F MORE EXAMPLES

To illustrate how the information-gain signal allocates credit on real trajectories, this section presents two step-by-step case studies drawn from the same ALFWorld training batch, both sampled for the same task instruction "put a cool pan in countertop". Standardization uses the batch statistics $\mu_G = 0.027$ and $\sigma_G = 0.032$ computed over 128 trajectories. For each case we show the generated plan, the executed actions with their per-step information gains IG_t, and the trajectory-level statistics: the outcome reward, the average gain $\bar{G}(\tau)$, its standardized value, and the planner coefficient α. For each case we also list the executor coefficients $\gamma_t = 1/\big(1 + e^{\mathrm{IG}_t}\big)$ computed from the logged gains.

F.1 CASE 1: A PLAN THAT GUIDES EXECUTION

The trajectory succeeds (outcome reward 10) with $\bar{G}(\tau) = 0.118$, standardized value 2.83, and planner coefficient $\alpha = 0.944$. The plan reads:

> *Step 1: Identify and locate the pan in the environment. Step 2: Retrieve the pan from its location. Step 3: Cool the pan using the fridge. Step 4: Locate a suitable countertop to place the cooled pan. Step 5: Place the cooled pan on the designated countertop.*

t	Action	IG_t	γ_t
0	(planning step)	0.000	–
1	go to diningtable 1	0.057	0.486
2	inventory	0.060	0.485
3	go to kitchen sink 1*	0.082	0.479
4	go to fridge 1	−0.005	0.501
5	open fridge 1	0.139	0.465
6	go to stoveburner 1	−0.003	0.501
7	go to stoveburner 2	−0.027	0.507
8	take pan 1 from stoveburner 2	0.152	0.462
9	go to fridge 1	0.641	0.345
10	cool pan 1 with fridge 1	0.118	0.470
11	go to countertop 1	0.034	0.491
12	move pan 1 to countertop 1	0.166	0.459

*The attempted action was not in the admissible set at that step. The planning step ($t = 0$) is scaled by α rather than γ_t.

The gains concentrate precisely on the plan-critical steps. The largest appears at $t = 9$ (`go to fridge 1`, $\mathrm{IG}_t = 0.641$): the plan explicitly instructs cooling the pan with the fridge, so the executor is markedly more confident in this move with the plan in context than without it. The next-largest gains fall on `move pan 1 to countertop 1` (0.166), `take pan 1 from stoveburner 2` (0.152), and `open fridge 1` (0.139), while exploratory navigation ($t = 4, 6, 7$) receives near-zero or negative gains. The mapping to coefficients makes this concrete: the plan-dictated $t = 9$ step receives the smallest coefficient ($\gamma_9 = 0.345$) and cedes most of its step-level credit to the planner, while the near-zero-gain navigation steps retain coefficients around 0.5; at the trajectory level, the high average gain yields $\alpha = 0.944$, assigning the planner nearly the full share of the planning-step advantage.

F.2 CASE 2: A PLAN THAT MISLEADS EXECUTION

The trajectory fails (outcome reward 0) with $\bar{G}(\tau) = 0.062$, standardized value 1.09, and $\alpha = 0.748$. The plan contains a self-contradictory instruction in its third step:

Step 1: Identify where to find a pan by searching cabinets systematically. Step 2: Retrieve the pan from its location. Step 3: Heat the pan if necessary to cool it. Step 4: Pick up the cooled pan. Step 5: Locate a countertop where the pan can be placed. Step 6: Place the cooled pan on the chosen countertop.

t	Action	IG_t	γ_t
0	(planning step)	0.000	–
1	go to cabinet 1	0.083	0.479
2	open cabinet 1	0.146	0.464
3	go to countertop 1	-0.028	0.507
4	go to diningtable 1	-0.075	0.519
5	go to stoveburner 1	-0.013	0.503
6	go to sinkbasin 1	-0.071	0.518
7	go to stoveburner 2	0.076	0.481
8	take pan 1 from stoveburner 2	0.116	0.471
9	move pan 1 to countertop 1	0.204	0.449
10	go to fridge 1	0.047	0.488
11	cool pan 1 with fridge 1*	0.089	0.478
12	open fridge 1	0.027	0.493
13	go to microwave 1	0.124	0.469
14	cool pan 1 with microwave 1*	-0.039	0.510
15	go to fridge 1	0.028	0.493
16	go to microwave 1	0.056	0.486
17	heat pan 1 with microwave 1*	0.192	0.452
18	(malformed output: planning-only format)	0.061	0.485
steps 19–29: repeated navigation without completing the task (omitted)			

*The attempted action was not in the admissible set at that step. The planning step ($t = 0$) is scaled by α rather than γ_t. The confusion in Step 3 ("Heat the pan if necessary to cool it") propagates measurably into execution. The largest gain of the trajectory falls on the premature `move pan 1 to countertop 1` at $t = 9$ (0.204), which places the pan before any cooling—an ordering the muddled plan encourages—after which the executor no longer holds the pan, so its subsequent cooling attempts ($t = 11, 14$) are not admissible. The plan-induced `heat pan 1 with microwave 1` at $t = 17$ likewise receives one of the largest gains (0.192): the plan is actively steering the executor in the wrong direction. Because the trajectory fails while the plan demonstrably influenced execution ($\bar{G}$ above the batch mean), the planner retains a substantial share of the penalty ($\alpha = 0.748$). The step-level modulation acts symmetrically on this failed trajectory: with $A_{\text{episode}} < 0$, the plan-misled decisions at $t = 9$ and $t = 17$ receive the smallest coefficients ($\gamma_9 = 0.449$, $\gamma_{17} = 0.452$), so the executor's penalty is attenuated precisely on the steps the plan dictated, whereas near-autonomous steps ($\gamma_t \approx 0.5$) keep a larger share of the failure signal. Blame for the plan-driven mistakes is thereby shifted toward the planner through α rather than charged to the executor.