LEAH: JOINTLY TRAINING LLM AGENTS TO ACT AND REVISE THEIR HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents depend on both the model and its harness, which shapes context, tool use, and interaction with the environment. Useful harness guidance changes as execution unfolds and depends on the model's ability to use it, yet existing approaches often optimize one component while fixing the other. We propose LEAH, which jointly trains an agent to decide when and how to revise its harness during execution and to act under the resulting harness toward a shared task objective. Since revisions affect task outcomes through subsequent execution, joint learning requires distinguishing their contributions. LEAH uses hierarchical reinforcement learning with separate advantage estimates: revisions receive credit for downstream execution, while actions are evaluated relative to their harness context. LEAH achieves a 96.43% success rate on ALFWorld, a task goal completion score of 65.48 on AppWorld Test-Normal, and 76.04% on MedAgentGym, outperforming execution-only and flat joint-training baselines. Further analyses show that revising the harness during execution can improve performance without updating model weights. During joint training, revisions become more relevant and actionable, and the agent makes more task progress after revising its harness.

1 INTRODUCTION

Large Language Model (LLM) agent systems are increasingly capable of carrying out complex tasks, from daily assistance to scientific problem solving. Such a system consists of two central components: the underlying LLM and its **harness**, the surrounding configuration that determines how the model maintains context, uses tools, and interacts with its environment. An agent's behavior is thus shaped not only by the model's capabilities but also by the harness through which those capabilities are expressed, and the same model can perform very differently under different harnesses. This dependence has made the harness itself an important target of optimization: recent work on harness and context engineering has improved agent performance by refining prompts, skills, tool interfaces, or the surrounding runtime (Zhang et al., 2026; Lee et al., 2026a; Chen et al., 2026; Yang et al., 2026).

Optimizing the harness around a given model supports modular development: the harness can be inspected, reused, or replaced without modifying the model. In this work, we take a *complementary* view, treating the agent as an *integrated system* in which the model and harness jointly shape task performance. From this perspective, our goal is to improve task performance by improving both the support provided by the harness and the model's ability to use that support effectively.

For agentic tasks, this interaction between model and harness must be considered as execution unfolds. An agent may encounter an unexpected tool failure, discover that a previously useful procedure no longer applies, or identify a better strategy through interaction. These developments can change what harness guidance is useful to the model. Harness specified in advance may not address the situations encountered during execution, while attempting to cover every possibility risks introducing irrelevant context that can distract the model (Yang et al., 2025). This motivates allowing the harness to be revised using execution feedback, a direction explored in recent work (Karten et al., 2026; Ning et al., 2026). We consider an agent that makes these revisions as part of task execution: it uses its interaction history to decide whether and how to revise its active harness guidance, then continues acting under the resulting harness.

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.

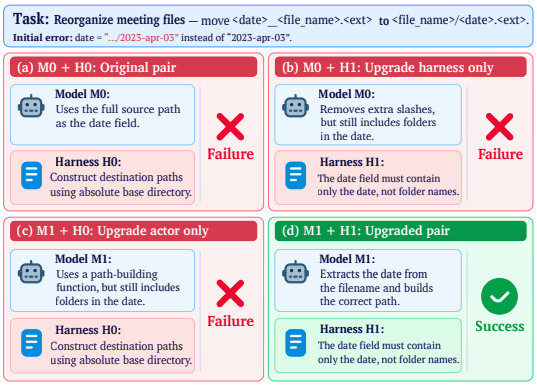


Figure 1: Illustrative example showing model–harness dependence. The original model and harness fail because the date is extracted from the full path. Updating only the model or only the harness leaves the underlying mismatch unresolved. The updated pair succeeds by using the revised guidance to extract the filename fields correctly.

Making harness revisions available allows the agent to adjust its guidance to the unfolding trajectory, but leaves open how to learn to make and use those revisions effectively. Crucially, the value of a revision depends on how the model interprets the resulting guidance and translates it into subsequent actions. This dependence also matters during training, because as the model's execution capabilities and strategies improve, the guidance it benefits from may change. Conversely, changes in the agent's revision behavior alter the harnesses under which it acts and learns. Guidance that is effective for one version of the model may therefore become less effective after a policy update (Yu et al., 2026). Responsiveness within an episode does not, on its own, ensure that revisions remain useful as execution behavior changes during training.

Existing work has largely approached the two sides separately: harness-engineering methods improve the harness around a given model (Zhang et al., 2026; Chen et al., 2026; Lee et al., 2026a), while agent RL typically improves execution under a prescribed harness (Chen et al., 2025; Jiang et al., 2026; Peng et al., 2026). Neither direction, by itself, addresses how revision and execution should be learned as coupled behaviors. These interdependencies motivate our central learning question: *how should an agent learn to revise its harness and to act effectively under the revisions it makes at the same time?*

Consider a digital-assistance task in which an agent must reorganize meeting files. For example, the absolute source path `.../work/meetings_files/2023-apr-03__slides.ppt` should be moved to `.../work/meetings_files/slides/2023-apr-03.ppt`. This requires extracting the date and document name from the filename rather than from the full path. The original model incorrectly includes directory names in the date field, producing a malformed destination. Updating only the harness, which points out the problem "date field incorrectly contains folder names", does not fix the problem, the original model cannot reliably follow the guidance and merely performs superficial path cleanup. Updating only the model also fails, although it adopts a better path-construction operation, it still uses the wrongly extracted date under the original guidance. Only the updated pair succeeds, with the model using the revised guidance to isolate the filename, extract its fields, and construct the correct destination. Figure 1 summarizes these cases.

Motivated by these observations, we propose **LEAH** (LEarn to Act and Harness), a framework for **jointly training an LLM agent to act and revise its harness**. In LEAH, harness revision is a learned component of the agent's decision process. The agent learns when and how to revise its harness using the interaction history, together with how to act effectively under the resulting harness. Both behaviors are optimized toward the same task objective: revision decisions are learned through their contribution to downstream execution, while execution is learned under harnesses produced by the agent's own revisions. We target settings where useful guidance emerges during interaction and its successful use depends on the model's learned execution capabilities. By training both behaviors, we seek to improve task performance in cases that require not only more appropriate guidance, but also a better ability to act on that guidance.

Joint learning of harness revision and execution in LEAH introduces a delayed credit-assignment problem: the value of a harness revision is realized only through the subsequent behavior it induces. Indeed, Sec. 5.2 shows that directly applying standard RL offers limited gains over training with a static base harness. This interaction naturally forms a hierarchy, since each revision persists and shapes a sequence of later environment actions. We therefore train LEAH with **hierarchical reinforcement learning** (Figure 2), using separate advantages for harness revisions and environment actions. Revisions are optimized according to their downstream effects, while actions are optimized within the harness context in which they are taken.

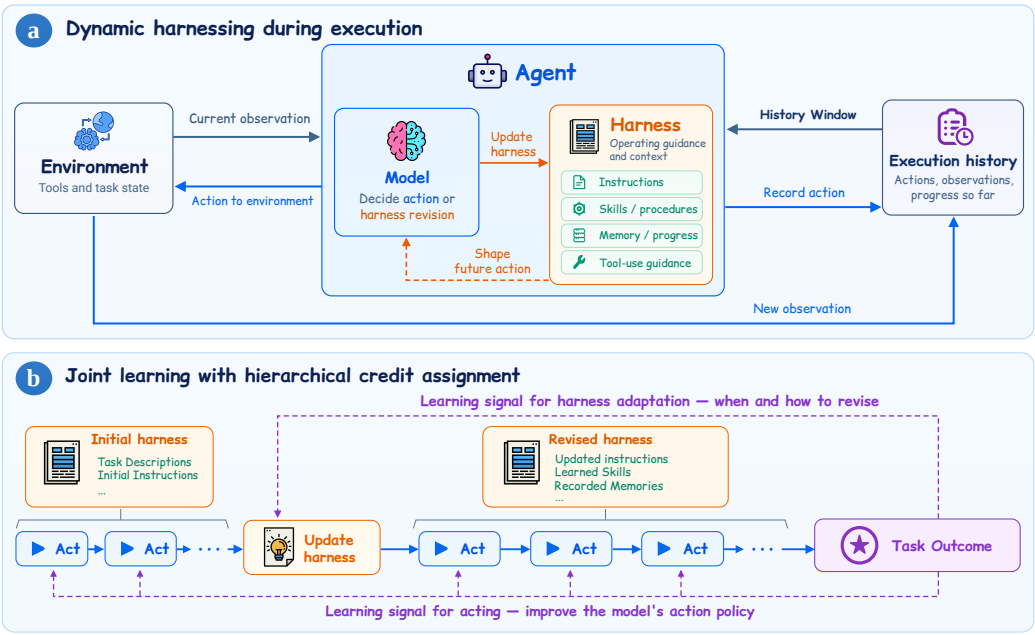


Figure 2: Overview of LEAH. (a) During execution, the agent can either act in the environment or revise its episode-local harness based on the interaction history; the revised harness then conditions subsequent actions. (b) LEAH jointly learns harness revision and execution with hierarchical credit assignment: revisions receive credit through the downstream behavior they induce, while environment actions are optimized relative to the harness context under which they are taken.

Empirically, we evaluate LEAH on three interactive agent environments spanning household tasks, digital assistance, and medical reasoning: ALFWorld, AppWorld, and MedAgentGym. LEAH achieves a task success rate of 96.43% on ALFWorld and a task goal completion score of 65.48 on AppWorld. On MedAgentGym, it achieves 76.04%, compared with 71.88% for the strongest RL trained baseline. Beyond final performance, our experiments isolate the roles of harness revision and joint learning. Allowing harness revisions improves performance even with frozen model weights, showing that execution-time revision itself can be useful. When revision and execution are learned jointly, however, standard flat RL provides only limited improvement over training against a fixed base harness, whereas explicitly exploiting the hierarchical structure yields substantially stronger gains. Further analyses examine how revision behavior and execution behavior evolve together during training.

We discuss related work in Appendix B. We summarize our main contributions as follows:

- **Joint learning of harness revision and execution.** We introduce LEAH, a framework that trains the LLM agent to decide when and how to revise its harness during task execution and to act effectively under the resulting harness, optimizing both behaviors toward the same task objective.
- **Hierarchical credit assignment for joint learning.** We identify the hierarchical structure induced by persistent harness revisions and develop a reinforcement learning approach that assigns credit separately to harness-level revisions and actions taken under the resulting harness.
- **Empirical evaluation and analysis.** Across ALFWorld, AppWorld, and MedAgentGym, LEAH substantially improves performance over execution-only and flat joint-training baselines. Our analyses further characterize the benefit of execution-time harness revision and how revision and execution behaviors evolve together during training.

2 Background

2.1 Language-Model Agents and Harnesses

A language-model agent is a closed-loop system that interacts with an environment through a language model and its surrounding harness. Let $\mathcal{M}_\theta$ denote a language model with parameters θ, and let $\mathcal{H}$ denote an agent harness. The term *harness* has been used in a broad sense to encompass the mechanisms and guidance surrounding the model, including context construction, memory,

tools, reusable skills, and other runtime components (Karten et al., 2026; Lee et al., 2026a). In this work, we narrow the range of the term harness and focus on an editable subset of these components, specified in Sec. 3. Together, $\mathcal{M}_\theta$ and $\mathcal{H}$ induce an agent policy over environment actions.

2.2 Environment and Interaction

Let $x \sim p_{\mathcal{X}}$ denote a task drawn from a fixed task distribution. Each task defines a finite-horizon partially observable environment with action space $\mathcal{A}$, observation space $\mathcal{O}$, horizon T, and discount factor $\gamma \in (0, 1]$. At step t, the agent receives an observation $o_t \in \mathcal{O}$, selects an environment action $a_t \in \mathcal{A}$, and receives a task reward r_t together with the next observation o_{t+1}.

The agent's decisions at each step are conditioned on the observable history and task description

$$h_t = (x, o_0, a_0, o_1, \ldots, a_{t-1}, o_t)\,. \tag{1}$$

Observations include environment feedback such as tool outputs, execution results, and error messages. Under a fixed model and harness, the agent selects

$$a_t \sim \pi_\theta(\cdot \mid h_t, \mathcal{H}). \tag{2}$$

We denote the resulting episode trajectory by

$$\xi = (o_0, a_0, r_0, o_1, \ldots, a_{T-1}, r_{T-1}, o_T)\,. \tag{3}$$

2.3 Model and Harness Optimization

Let $p_{\theta,\mathcal{H}}(\xi \mid x)$ denote the trajectory distribution induced by the agent policy and the task environment, including its initial conditions and dynamics. For an episode horizon T and discount factor γ, define the task return as

$$R_x(\xi) = \sum_{t=0}^{T-1} \gamma^t r_t, \tag{4}$$

where r_t is the task reward associated with the environment action at step t. The objective for the model–harness pair is

$$J(\theta, \mathcal{H}) = \mathbb{E}_{x\sim p_{\mathcal{X}},\ \xi\sim p_{\theta,\mathcal{H}}(\cdot|x)}\left[R_x(\xi)\right]. \tag{5}$$

Model optimization under this objective varies θ while holding the harness specification fixed, maximizing $J(\theta, \bar{\mathcal{H}})$. Conversely, harness optimization varies $\mathcal{H}$ while holding the model fixed, maximizing $J(\bar{\theta}, \mathcal{H})$. Here, $\bar{\theta}$ and $\bar{\mathcal{H}}$ denote fixed choices of the corresponding components. From an integrated-system perspective, a natural objective is to seek the model–harness pair with the highest task performance:

$$\max_{\theta,\mathcal{H}} J(\theta, \mathcal{H}). \tag{6}$$

Directly optimizing this objective is challenging because θ and $\mathcal{H}$ are fundamentally different optimization objects: θ consists of numerical model parameters, whereas $\mathcal{H}$ is typically expressed through structured discrete or textual specifications such as instructions, skills, and code snippets. Thus, model and harness optimization do not naturally share a common update mechanism.

More importantly, the formulation above treats $\mathcal{H}$ as fixed throughout an episode. As motivated in Sec. 1, the harness guidance useful to an agent can change as execution unfolds. We therefore extend the formulation by allowing the agent to revise its harness during execution based on the interaction history. We next formalize this execution-time harness revision.

3 Harness Revision During Execution

We generalize the fixed-harness formulation by allowing the agent to revise an episode-local harness during execution. Each episode begins from a common base harness and an empty editable state. At each step, the agent may leave the editable state unchanged or revise it based on the interaction history, then act under the resulting harness. Revisions persist within the episode but are reset between episodes; what transfers across tasks is the learned revision behavior rather than the edited content itself.

Let H_{base} denote the fixed harness and c_t the episode-local editable content, with $c_0 = \emptyset$. The active harness is $H_t = H_{\text{base}} \oplus c_t$, where $\oplus$ denotes deterministic composition.

Formally, we decompose the harness into a fixed base component and an episode-local editable component. Let $\mathcal{H}_{\text{base}}$ denote the fixed portion of the harness, and let c_t denote the editable content active at step t. We initialize $c_0 = \varnothing$ at the beginning of each episode. The effective harness available to the agent at step t is $\mathcal{H}_t = \mathcal{H}_{\text{base}} \oplus c_t$, where $\oplus$ denotes a deterministic composition operator that incorporates the editable content into the base harness.

In our implementation, c_t contains textual skills, memory, and additional instructions; revisions affect only this guidance and do not modify the environment, action semantics, reward, or success criteria. Specific editable harness structure is provided in Fig. 6.

We restrict each harness revision to a single atomic edit. Let $\mathcal{E} = \{\text{ADD}, \text{EDIT}, \text{DELETE}\}$ denote the set of admissible edit operations. Each revision targets at most one entry in the editable harness content: it may add a new entry, modify an existing entry, or delete an existing entry. Thus, a single revision cannot simultaneously modify multiple harness entries.

Harness revision and task execution are coupled through the active harness state. A revision changes the context under which subsequent environment actions are generated. Conversely, feedback produced during execution provides evidence about whether the current harness remains useful and how it might be revised. Revision and execution therefore form a closed feedback loop.

To formalize this interaction, let $\mathcal{Z}$ denote the space of admissible textual revisions. Each $z_t \in \mathcal{Z}$ specifies one atomic edit to the editable harness content: it may ADD a new entry, EDIT an existing entry, or DELETE an existing entry, together with the textual content required by that operation. We additionally introduce a distinguished null revision $\perp \notin \mathcal{Z}$ and define $\overline{\mathcal{Z}} = \mathcal{Z} \cup \{\perp\}$, where $z_t = \perp$ indicates that the current harness is left unchanged.

Within the agent, the harness-revision policy $z_t \sim \pi_\phi^{\text{rev}}(\cdot \mid h_t, \mathcal{H}_t)$, where $z_t \in \overline{\mathcal{Z}}$, determines whether and how to revise the current harness. Here, ϕ denotes the parameters governing harness revision.

Let $c_t^+ = U(c_t, z_t)$ denote the editable content after applying revision z_t, where U is the corresponding deterministic update operator and $U(c_t, \perp) = c_t$. The resulting harness is

$$\mathcal{H}_t^+ = \mathcal{H}_{\text{base}} \oplus c_t^+. \tag{7}$$

The execution policy then selects an environment action conditioned on the resulting harness,

$$a_t \sim \pi_\theta^{\text{exec}}\left(\cdot \mid h_t, \mathcal{H}_t^+\right), \tag{8}$$

where θ denotes the parameters governing task execution. After the environment transition, the revised editable state persists to the next step:

$$c_{t+1} = c_t^+, \qquad \mathcal{H}_{t+1} = \mathcal{H}_{\text{base}} \oplus c_{t+1} = \mathcal{H}_t^+. \tag{9}$$

The two decision policies are therefore coupled through the active harness state. Within each step, the revision decision determines the context under which the environment action is generated:

$$p_{\theta,\phi}(z_t, a_t \mid h_t, \mathcal{H}_t) = \pi_\phi^{\text{rev}}(z_t \mid h_t, \mathcal{H}_t)\, \pi_\theta^{\text{exec}}\left(a_t \mid h_t, \mathcal{H}_t^+\right). \tag{10}$$

The revision and execution policies need not have disjoint parameterizations. In particular, they may share part or all of the same underlying model parameters while remaining distinct decision distributions through their conditioning contexts and output spaces. In our experiments in Sec. 5, we adopt a natural choice where both are instantiated by the same underlying LLM.

4 JOINTLY LEARNING TO ACT AND REVISE THE HARNESS

4.1 LEARNING OBJECTIVE

Harness revision and execution ultimately serve the same task objective. In particular, harness revision does not receive a separate task reward: a revision is useful only insofar as it changes subsequent execution in a way that improves task performance. Let $p_{\theta,\phi}(\xi \mid x)$ denote the environment-trajectory distribution induced jointly by π_ϕ^{rev} and π_θ^{exec}, together with the task environment. The learning objective remains the same as Eq. 5:

$$J(\theta, \phi) = \mathbb{E}_{x \sim p_{\mathcal{X}},\, \xi \sim p_{\theta,\phi}(\cdot \mid x)}\left[R_x(\xi)\right]. \tag{11}$$

This objective can in principle be optimized directly using standard policy-gradient methods such as PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024). The challenge lies in the different roles

that revision and execution play in generating task return. The value of a harness revision depends on the execution behavior that follows it: the same revision may be useful when the model can effectively exploit its guidance and ineffective otherwise. Conversely, execution is learned under harness states produced by the revision policy, so changes in revision behavior alter the contexts in which execution is trained.

Flat RL does not explicitly separate credit between harness revisions and environment actions. The same return may reflect revision quality, execution quality, or both, and this attribution can shift as the two behaviors co-evolve during training. Although PPO or GRPO can optimize the shared task objective, their learning signal therefore entangles the two decision types. As shown in Sections 5.2 and 5.3, flat GRPO yields weaker learning and less evidence of revision–execution co-adaptation.

This motivates exploiting the structure of the revision–execution process explicitly. We formulate joint learning as a hierarchical reinforcement learning problem and assign credit according to the distinct roles of revision and execution, while preserving their shared task objective. We next introduce the corresponding hierarchical credit-assignment mechanism.

4.2 Coupled Credit Assignment

We begin from the two decision points induced by each revision–execution step. A revision first transforms the current harness $\mathcal{H}_t$ into the resulting harness $\mathcal{H}_t^+$, after which the agent acts under the resulting context. This gives two points at which the expected downstream task return can be evaluated: before revision and after revision. We present the high-level idea of advantage estimation for each decision here and defer the full derivation to E.1.

Let $V^{\mathrm{rev}}(h_t, \mathcal{H}_t)$ denote the expected remaining task return before the revision decision, and let $V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+)$ denote the expected remaining return once the resulting harness has been established. We assign revision credit according to

$$A_t^{\mathrm{rev}} = V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+) - V^{\mathrm{rev}}(h_t, \mathcal{H}_t), \tag{12}$$

so that a revision decision is reinforced when it moves the agent to a more valuable execution context.

Execution credit is defined relative to the harness established by the revision decision. Once $\mathcal{H}_t^+$ is fixed, the environment action a_t affects the task environment, producing the immediate reward r_t and the next interaction history h_{t+1}. The system then arrives at the next pre-revision decision point $(h_{t+1}, \mathcal{H}_{t+1})$. We use the following one-step execution credit signal

$$A_t^{\mathrm{exec}} = r_t + \gamma V^{\mathrm{rev}}(h_{t+1}, \mathcal{H}_{t+1}) - V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+). \tag{13}$$

to measure whether the selected environment action yields greater downstream task value than expected under the harness context established by the revision decision.

The two credit signals are themselves coupled. Revision credit is defined through the value of the execution context that a revision creates, while execution credit bootstraps from the value of the next revision decision point. This mirrors the underlying interaction: revision changes the conditions for execution, and execution determines the state from which the next revision decision is made.

4.3 Hierarchical Policy Optimization

Given the hierarchical credit signals above, the revision and execution policies can be optimized using their respective advantages. The resulting policy gradient takes the form

$$\nabla J(\theta, \phi) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_\phi \log \pi_\phi^{\mathrm{rev}}(z_t \mid h_t, \mathcal{H}_t) A_t^{\mathrm{rev}} + \nabla_\theta \log \pi_\theta^{\mathrm{exec}}(a_t \mid h_t, \mathcal{H}_t^+) A_t^{\mathrm{exec}}\right]. \tag{14}$$

When the two policies share underlying language-model parameters, the shared parameters receive gradient contributions from both terms in Eq. 14. This allows a single model to jointly improve how it revises its harness and how it acts under the resulting harness, while the role-specific advantages preserve distinct credit for the two types of decisions.

In practice, we estimate the value baselines $V^{\mathrm{rev}}(h_t, \mathcal{H}_t)$ and $V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+)$ using critic models, and optimize Eq. 14 using a PPO-style clipped objective with generalized advantage estimation (Schulman et al., 2015). We defer the policy-gradient derivation to Appendix E.2, full advantage estimator, PPO-style objective, critic implementation, and training algorithm to Appendix D.

Table 1: **Performance of LEAH**. We report success rate on validation set for ALFWorld, Task-Goal-Completion (TGC) on test-normal and test-challenge for AppWorld, and category-wise and average success rates for MedAgentGym. The upper panel evaluates frozen base models at inference time, the middle panel evaluates closed-source models, and the lower panel compares performance after training. "–" denotes not applicable or not evaluated.

Method	ALFWorld	AppWorld(%)		MedAgentGym(%)			
	SR (%) ↑	Test-N ↑	Test-C ↑	MedCalcBench ↑	BioCoder ↑	NPowerAI ↑	Avg. ↑
Base Model (Frozen Weights)							
ReAct	41.42	22.02	11.75	31.25	18.75	28.13	26.04
Periodic Revision (T=3)	37.10	29.17	15.11	37.50	18.75	31.25	29.17
Periodic Revision (T=5)	42.90	28.57	15.59	34.38	21.88	31.25	29.17
Periodic Revision (T=10)	39.30	30.36	17.03	28.13	21.88	28.13	26.04
Periodic Revision (T=20)	42.14	29.76	17.27	–	–	–	–
On-Demand Revision	45.00	31.55	17.75	37.50	18.75	40.63	32.29
Closed-source Models							
GPT-5.4 (ReAct)	70.95	71.43	–	87.50	75.00	59.38	73.96
GPT-5.4 (On-Demand Revision)	78.57	73.21	–	90.63	75.00	62.50	76.04
GPT-5.6-Sol (ReAct)	<u>90.95</u>	67.86	–	87.50	**81.25**	62.50	**77.08**
GPT-5.6-Sol (On-Demand Revision)	90.71	**73.21**	–	<u>90.63</u>	78.13	59.38	76.04
RL Trained Agents							
ReAct + GRPO	88.57	58.33	30.22	81.25	53.13	<u>81.25</u>	71.88
On-Demand Revision + GRPO	90.00	61.90	<u>33.09</u>	84.38	53.13	71.88	69.79
LEAH	**96.43**	<u>65.48</u>	**35.97**	**90.63**	<u>56.25</u>	**81.25**	<u>76.04</u>

5 EXPERIMENTS

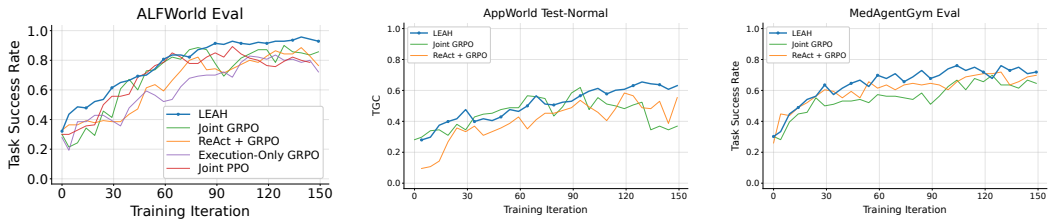
In this section, we present the empirical results of LEAH across three interactive agent environments spanning distinct domains: ALFWorld (Shridhar et al., 2021) for simulated household tasks, AppWorld (Trivedi et al., 2024) for digital assistance with applications and APIs, and MedAgentGym (Xu et al., 2026) for interactive medical reasoning and coding. Our experiments are organized around three questions. First, we ask whether allowing harness revision during execution, as formulated in Sec. 3, improves agent performance at inference time. Second, we study whether jointly learning harness revision and execution further improves performance over inference-time revision alone. Third, we examine whether hierarchical credit assignment improves joint learning over standard flat RL and leads to better coordination between revision and execution.

5.1 EXPERIMENT SETUP

Tasks. We train and evaluate LEAH separately on ALFWorld, AppWorld, and MedAgentGym. For ALFWorld, we follow the training and evaluation splits of Feng et al. (2026). For AppWorld, we use the training split provided by the environment for training and evaluate on the Test-Normal (Test-N) and Test-Challenge (Test-C) task sets. For MedAgentGym, due to the scale and computational cost of the full benchmark, we focus on three task categories: MedCalcBench, BioCoder, and NPowerAI. We jointly train a single agent on the combined training sets from these three categories and evaluate it on sampled subsets from the corresponding evaluation sets.

Baselines. We compare LEAH against both inference-time and learning-based baselines. For inference-time evaluation with frozen model weights, we consider *ReAct*, which uses a fixed minimal harness throughout execution, and *Periodic Revision*, which invokes a harness revision every T environment steps. We evaluate periodic schedules with $T \in \{3, 5, 10, 20\}$ and compare them with *On-Demand Revision*, corresponding to the execution-time revision mechanism introduced in Sec. 3, where the agent decides whether to revise its harness based on the execution trajectory. We also report the performance of strong closed-source models on these environments as reference.

For learning-based comparisons, we include *ReAct + GRPO*, which trains execution under a fixed minimal harness, and *Joint GRPO (On-Demand Revision)*, which jointly trains harness revision and execution using a standard flat RL objective. We compare them with LEAH, which uses hierarchical credit assignment to distinguish the contributions of harness revisions and environment actions.
Implementation. We use Qwen3.5-9B (Qwen Team, 2026) as the base model for ALFWorld and AppWorld, and Qwen2.5-7B-Instruct (Team, 2024) for MedAgentGym. Unless otherwise specified, harness revision and execution share the same underlying LLM parameters. Within each environment, we match the rollout and training budgets across learning-based methods whenever applicable. All controlled comparisons are initialized from the same base model and use the same training tasks,



(a) ALFWorld (b) AppWorld (c) MedAgentGym

Figure 3: **Learning curves on different environments.** We compare ReAct + GRPO, On-Demand Revision + GRPO, and LEAH with hierarchical credit assignment. On ALFWorld, we additionally include the execution-only variant to isolate the effect of jointly optimizing harness adaptation.

evaluation protocol, and training hyperparameters, provided in Appendix C.1. The prompt templates for ReAct and On-Demand Harness Revision are given in Appendix C.2. Additional implementation details are provided in Appendix C.

5.2 Experiment Results

RQ1: Does harness revision during execution improve inference-time agent performance? With frozen model weights, On-Demand Revision improves over ReAct by 3.6 pp on ALFWorld, 9.5 pp on AppWorld Test-N, and 6.25 pp on MedAgentGym (Table 1). On ALFWorld, it also outperforms all tested periodic schedules, indicating that model-controlled revision can be more effective than revision on a fixed schedule.

RQ2: Should harness revision and execution be learned jointly? We next investigate the effect of jointly training harness revision and execution, compared with training execution while keeping the model used for harness revision fixed. Table 2 presents a controlled comparison on ALFWorld, where all variants use the same base model and GRPO training setup. To isolate the effect of joint training, we separately parameterize harness revision and execution in this ablation, allowing the revision model to remain frozen while the execution model is trained. Training only the execution policy while keeping the harness-revision model frozen achieves 83.57%, below the 88.57% achieved by ReAct. The learning curves in Fig. 3a further suggest that this setting is less stable than training ReAct. In contrast, jointly training harness revision and execution improves success to 90.00%, a gain of 6.43 pp over the variant with a frozen harness-revision model.

Table 2: **Joint optimization of acting and harness adaptation.** All variants use the same model and GRPO training setup on ALFWorld.

Setting	Optimized Decisions	Success (%)
ReAct	Act	88.57%
On-demand Revision	Act	83.57%
On-demand Revision	Act + Revision	90.00%

RQ3: Does Hierarchical Credit Assignment Improve Joint Learning? Finally, we study whether explicitly accounting for the hierarchical structure of the revision–execution process improves learning. We first compare Joint GRPO with LEAH. As shown in the lower panel of Table 1, LEAH achieves 96.4% on ALFWorld, compared with 90.0% under flat GRPO, with similar improvements on AppWorld and MedAgentGym. To separate the effect of hierarchical credit assignment from the choice of RL algorithm, we additionally train the same revision–execution architecture with standard PPO. As shown in Fig. 3a, LEAH also outperforms flat PPO under the same architecture and training setup, suggesting that the improvement comes from the hierarchical credit assignment rather than from the choice of optimizer alone. The learning curves further show that LEAH learns faster and more stably across environments and converges to higher final performance than the flat baselines. Notably, LEAH is also competitive with substantially stronger closed-source models. These results suggest that jointly learning harness revision and execution with hierarchical credit assignment can substantially narrow, and in some settings overcome, the performance gap between smaller open-source agents and stronger closed-source models.

To examine the behaviors learned by LEAH and whether harness revision and execution become increasingly coordinated during training, we further analyze rollout trajectories across training iterations in Sec. 5.3.

5.3 Co-Adaptation Analysis

To characterize how revision and execution evolve during learning, we analyze ALFWorld rollouts from LEAH and Joint GRPO. Figure 4 tracks four measures: first-revision timing, success among revision-containing episodes, revision quality, and post-revision milestone completion.

Concretely, we analyze three aspects of this interaction: *when* the agent chooses to revise its harness, *how* the quality of its revisions changes during training, and *how* execution progresses following a revision. Together, these analyses probe for behavioral signatures of increasing coordination between harness revision and execution.

Figure 4 summarizes four behavioral measures tracked throughout training: (a) the median step of the first harness revision, (b) task success among episodes containing at least one revision, (c) the fraction of revision content judged to be qualified by an LLM evaluator, and (d) the fraction of eligible task milestones completed within five environment actions after the first revision.

When to revise. In Fig. 4(a), the first harness revision invocation under LEAH gradually shifts toward earlier stages of execution, whereas its timing under Joint GRPO remains substantially more variable. Earlier revision alone is not necessarily desirable; its value depends on whether the resulting harness supports successful execution. Fig. 4(b) moreover reports task success among episodes containing at least one harness revision. Under LEAH, success among these episodes rises markedly over training, while Joint GRPO shows no comparable improvement. Taken together, these trends suggest LEAH increasingly integrates harness revision into successful execution: revisions occur earlier, while episodes containing revisions become more productive.

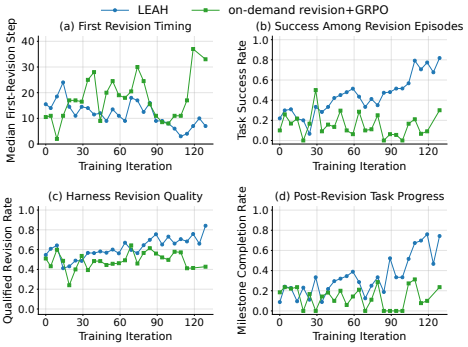


Figure 4: Behavioral trends of harness–execution co-adaptation in training

How to revise. Learning when to revise is useful only if the agent also learns what changes to make to its harness. Fig. 4(c) tracks the fraction of revision content judged as qualified by an LLM-as-Judge, using the evaluation rubric provided in Appendix C.3. A revision is considered qualified when its content is supported by the trajectory available before the revision, relevant to the current task, and actionable. The judge is blinded to subsequent execution and task outcomes. Under LEAH, the fraction of qualified revisions increases over training, whereas Joint GRPO does not exhibit a comparable sustained improvement. This suggests that hierarchical credit assignment improves not only when the agent revises its harness, but also the quality of the guidance it constructs from the execution context.

How to act after revision. Finally, we examine whether execution becomes more productive after a harness revision. Fig. 4(d) reports the fraction of eligible task milestones completed within five environment actions following the first harness revision, pooling target-object pickup and required transformation milestones. A milestone is considered eligible only if it had not already been completed before the revision. Under LEAH, post-revision milestone completion improves steadily over training, whereas Joint GRPO shows no comparable sustained trend. This provides action-level evidence that execution becomes increasingly effective at making concrete task progress under the harness contexts produced by revision, complementing the revision-quality analysis in Fig. 4(c). Together, these trends are consistent with increasing coordination between harness revision and execution under LEAH, a pattern not observed under Joint GRPO. We also provide qualitative examples in Appendix F.

6 Conclusion

We introduce LEAH, a framework that jointly trains an LLM agent to act and revise its harness during execution. LEAH treats harness revision as part of the agent's decision process and uses hierarchical credit assignment to separate revision and execution credit. Across ALFWorld, AppWorld, and MedAgentGym, LEAH outperforms execution-only and flat joint-training baselines, with analyses showing increasing coordination between revision and execution during training. These results support learning the model and harness as coupled components of a single agent system.

7 AI STATEMENT

AI was used to polish paper writing, for retrieval of related work, and for implementation of scaffolding code. All texts and code are verified and proof-read by authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

8 ETHICS STATEMENT

This work does not involve human subjects, user studies, or the collection of personal data. All experiments use publicly available benchmarks under their respective licenses. Closed-source models were accessed only for evaluation, through their official APIs and in accordance with the providers' terms of use.

Agents that revise their own harness raise the concern that revisions could bypass intended constraints or induce unintended behavior. In LEAH, revisions are limited to an episode-local editable component (skills, memory, and additional instructions). The base harness is fixed, and revisions cannot alter the environment, tools, action space, reward, or success criteria. Every revision is logged as explicit text that can be inspected, which makes adaptation easier to audit than changes encoded only in model weights. Optimizing revisions for task reward could nonetheless exploit weaknesses in benchmark success checkers, and in real-world deployments learned revisions could lead to unintended actions. We therefore recommend keeping safety-critical constraints in the fixed base harness, auditing revisions, and requiring human confirmation for irreversible actions.

9 REPRODUCIBILITY STATEMENT

Appendix C documents the experimental setup, including models, rollout and training budgets, optimization and sampling hyperparameters, evaluation protocols, harness-revision constraints, prompt templates, and the LLM-judge rubric. Controlled comparisons use matched base models, training tasks, evaluation protocols, and training budgets whenever applicable. We plan to release the code in the near future.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture, 2016. URL https://arxiv.org/abs/1609.05140.

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.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026.

Mingyue Cheng, Shuo Yu, Daoyu Wang, Qingchuan Li, Xiaoyu Tao, Jie Ouyang, Yucong Luo, Yitong Zhou, Qi Liu, and Enhong Chen. Agent-r1: A unified and modular framework for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2511.14460.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition, 1999. URL https://arxiv.org/abs/cs/9905014.

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, 2026.

Jian Hu, Yifan Zhang, Hao Zhang, Binfeng Xu, Shaokun Zhang, Hongqing Peng, Zhiding Yu, Pavlo Molchanov, Jan Kautz, and Yi Dong. Flashreinforce: Critic-free single-rollout asynchronous rl for agentic language models. 2026. URL https://github.com/yifanzhang-pro/FlashREINFORCE.

Pengcheng Jiang, Zhiyi Shi, Kelly Hong, Xueqiang Xu, Jiashuo Sun, Jimeng Sun, Hammad Bashir, and Jiawei Han. Harness-1: Reinforcement learning for search agents with state-externalizing harnesses. *arXiv preprint arXiv:2606.02373*, 2026.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026a.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026b. URL https://openreview.net/forum?id=tmbOUyFx3R.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.03680.

Xuying Ning, Dongqi Fu, Tianxin Wei, Hanqing Zeng, Yuanchen Bei, Bingxuan Li, Zihao Li, Qifan Wang, Xiang Shen, Yifan Wu, et al. Evoharness-rl: Learning self-evolving runtime harness for long-horizon llm agents. *arXiv preprint arXiv:2608.05446*, 2026.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. HiPER: Hierarchical plan–execute RL for multi-turn LLM agents. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=RJWQcgd3hX.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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 Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

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 *ACL*, 2024.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017. URL https://arxiv.org/abs/1703.01161.

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, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2504.20073.

Ran Xu, Yuchen Zhuang, Yishan Zhong, Yue Yu, Zifeng Wang, Xiangru Tang, Hang Wu, May Dongmei Wang, Peifeng Ruan, Donghan Yang, et al. Medagentgym: A scalable agentic training environment for code-centric reasoning in biomedical data science. In *International Conference on Learning Representations*, volume 2026, pp. 89688–89730, 2026.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Wang, and Liangming Pan. How is llm reasoning distracted by irrelevant context? an analysis using a controlled benchmark, 2025. URL https://arxiv.org/abs/2505.18761.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Zhou Yu, Bin Bi, Shiva Kumar Pentyala, Shubham Mehrotra, Sougata Chaudhuri, Shilpa Bhagavath, Zeyuan Chen, Ran Xu, Phil Mui, James Zhu, et al. Co-evolving harnesses and models: On-policy correction helps weaker models catch up where imitation fails. *arXiv preprint arXiv:2609.09134*, 2026.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl, 2024. URL https://arxiv.org/abs/2402.19446.

A Limitations

Our study has several limitations. First, LEAH currently revises only textual, episode-local harness content, including memory, skills, and additional instructions. Revisions are reset between episodes, so we do not study persistent harness evolution or whether useful revisions can transfer across tasks. Second, although we evaluate across three interactive environments, our experiments cover a limited set of model scales and task domains; in particular, the MedAgentGym evaluation uses three selected task categories rather than the full benchmark. Extending LEAH to persistent cross-episode adaptation, broader harness components, and larger-scale environments and models remains an important direction for future work.

B Related Work

Harness engineering and adaptation. A growing line of work treats the harness surrounding an LLM as an optimization target rather than a fixed implementation detail. Agentic Context Engineering (ACE) evolves contextual guidance from execution experience, while methods such as SkillOpt optimize reusable skills and procedures available to the agent (Zhang et al., 2026; Yang et al., 2026). Other work targets broader parts of the agent runtime. Meta-Harness uses an outer-loop coding agent to search over executable harnesses using accumulated evaluation results and execution traces (Lee et al., 2026b), while methods for diagnosing and repairing harness failures use failed trajectories to identify deficiencies in the surrounding agent system (Chen et al., 2026). Continual Harness and EvoHarness-RL further study harness adaptation from online execution experience (Karten et al., 2026; Ning et al., 2026).

Reinforcement learning for LLM agents. Reinforcement learning has become an increasingly important approach for training LLM agents on interactive and long-horizon tasks, where models must learn from delayed task outcomes while repeatedly incorporating environment feedback. Recent work develops both algorithmic and systems support for this setting, including memory-efficient PPO variants for stateful digital agents (Chen et al., 2025), group-based policy optimization for multi-turn agents (Feng et al., 2026), and RL methods that train agents together with state-externalizing harness mechanisms (Jiang et al., 2026). RAGEN studies trajectory-level multi-turn RL and training instabilities through the StarPO framework (Wang et al., 2025), while Agent-R1 provides a step-level formulation and modular infrastructure for end-to-end agentic RL (Cheng et al., 2026). Agent Lightning further decouples agent execution from RL training and introduces explicit credit assignment for complex agent trajectories (Luo et al., 2025). Other recent approaches, such as FlashREINFORCE, explore critic-free and single-rollout policy-gradient training for agentic language models (Hu et al., 2026). Despite these advances, most agent-RL methods optimize model behavior under a prescribed interaction structure or harness. LEAH instead treats harness revision itself as part of the learned decision process, so task return must be attributed not only to environment actions but also to revisions that alter the context under which subsequent actions are generated.

Hierarchical reinforcement learning. Hierarchical reinforcement learning (HRL) addresses long-horizon decision making through temporal abstraction and multi-level control. Classical formulations include the options framework, which extends primitive actions to temporally extended policies (Sutton et al., 1999), and MAXQ, which decomposes an MDP and its value function into a hierarchy of subtasks (Dietterich, 1999). Later approaches learn hierarchical structure more directly: Option-Critic jointly learns intra-option policies and termination conditions (Bacon et al., 2016), while FeUdal Networks separate a slower high-level manager from a lower-level worker operating at the environment timescale (Vezhnevets et al., 2017). More recently, ArCHer applies hierarchical RL to multi-turn LLM agents, combining high-level value learning across turns with low-level token-level policy optimization to address delayed rewards and long-horizon credit assignment (Zhou et al., 2024).

LEAH exploits a different source of hierarchy: harness revisions establish a persistent context that conditions subsequent environment actions. Unlike conventional options or subgoals, a revision does not directly act in the environment or receive a separate task reward; its value is realized through the execution it induces. This motivates our coupled credit assignment between harness-revision and execution decisions.

C Implementation Details

In this section, we include implementation details omitted in the main text due to space.

C.1 Experiment Details

ALFWorld and AppWorld. We use Qwen3.5-9B as a shared parameterization of harness revision and execution policies and train both jointly with LEAH. Each generation uses a maximum context length of 2048 tokens (ALFWorld) / 4096 tokens (AppWorld), including up to 512 response tokens (ALFWorld) / 1024 response tokens (AppWorld). Episodes are limited to 50 environment steps. The agent receives the two most recent observation–action pairs and uses permissive admissible-action projection. We collect 128 trajectories per rollout iteration, with one sample per task, and train for 150 iterations, including five initial critic-only warmup iterations. The actor and critic learning rates are fixed at 10^{-6} and 10^{-5}, respectively. Rollout sampling uses temperature 0.7 and top-p 1.0, while evaluation uses temperature 0 and top-p 0.95. We evaluate on all 140 valid_seen evaluation tasks (ALFWorld) / all 168 test-normal tasks (AppWorld) every five iterations. The policy KL-loss coefficient is 0.01.

MedAgentGym. We use Qwen2.5-7B-Instruct as a shared parameterization of harness revision and execution policies and train both jointly with LEAH. Each generation uses a maximum context length of 2048 tokens, including up to 512 response tokens. Episodes are limited to 15 environment steps. The agent receives the two most recent observation–action pairs and uses permissive admissible-action projection. We collect 128 trajectories per rollout iteration, with one sample per

task, and train for 150 iterations, including five initial critic-only warmup iterations. The actor and critic learning rates are fixed at 10^{-6} and 10^{-5}, respectively. Rollout sampling uses temperature 0.7 and top-p 1.0, while evaluation uses temperature 0 and top-p 0.95. We evaluate on all 96 evaluation tasks every five iterations, which are sampled equally from the three selected task categories, MedCalcBench, BioCoder and NPowerAI. The policy KL-loss coefficient is 0.01.

C.2 Prompt Templates

Harness Revision Prompt Template for ALFWorld

You are an agent completing a task in the ALFWorld environment.
You are now deciding whether and how to revise your own harness before taking the next environment action.
Your harness contains three types of editable operating guidance:
- Memory: task-relevant facts supported by observed interaction.
- Skills: procedures that guide task execution.
- Additional instructions: guidance for subsequent decisions.

You may leave the harness unchanged or apply exactly one atomic edit to one entry. A revision may preserve useful information, improve a procedure, or adjust guidance based on the task and the observed interaction. A revision is not required at every step.
Choose `NO_CHANGE` when the current harness does not need an edit. Otherwise, choose one `ADD`, `EDIT`, or `DELETE` operation.
Constraints:
- Use only the task description, fixed base instructions, current observation, interaction history, and current harness provided.
- Do not invent observations, completed actions, or object states.
- Do not modify the fixed base instructions, environment, tools, action semantics, or task success criteria.
- Do not execute an environment action.
- Do not request another revision decision.
- Do not modify more than one entry or return a replacement for the entire harness.
- `EDIT` and `DELETE` must reference an existing entry ID.
- For `ADD`, the runtime will assign the new entry ID.

Return exactly one JSON object and no other text.
Allowed output forms:

```
{"op": "NO_CHANGE"}
{"op": "ADD", "section": "<section>", "content": "<new entry>"}
{"op": "EDIT", "section": "<section>", "id": "<existing ID>",
  "content": "<replacement entry>"}
{"op": "DELETE", "section": "<section>", "id": "<existing ID>"}
```

Replace `<section>` with exactly one of: `"memory"`, `"skills"`, or `"additional_instructions"`.
Fixed base instructions:
{base_instructions}
Task:
{task_description}
Environment actions already executed:
{step_count}
Interaction history:
{interaction_history}
Current observation:
{current_observation}
Current admissible actions:
{admissible_actions}
Current editable harness:
Memory:
{memory_entries}
Skills:
{skill_entries}
Additional instructions:
{instruction_entries}
Decide whether to leave the harness unchanged or apply one atomic edit before the next environment action.

Figure 5: Harness revision prompt template for ALFWorld. Given the current trajectory and editable harness state, the agent may leave the harness unchanged or apply one atomic edit before taking the next environment action.

Environment Action Prompt Template for ALFWorld

You are an agent completing a task in the ALFWorld environment.
The harness-revision decision for this environment step is complete. Your task now is to choose exactly one environment action using the current observation, interaction history, and active harness.
The active harness contains memory, skills, and additional instructions that may support task execution. Treat these entries as operating guidance, not as evidence that an action has occurred or that a task condition has been satisfied.
Current environment feedback and the admissible-action list take precedence over conflicting harness entries.
Constraints:
- Choose exactly one action appearing verbatim in the current admissible-action list.
- Do not request, propose, or apply a harness revision in this response.
- Do not treat an attempted action as a confirmed state change.
- Do not claim task completion without the environment's success signal.

Return only the selected environment action in this format:
<action>exact admissible action</action>
Fixed base instructions:
{base_instructions}
Task:
{task_description}
Environment actions already executed:
{step_count}
Interaction history:
{interaction_history}
Current observation:
{current_observation}
Current admissible actions:
{admissible_actions}
Active editable harness:
Memory:
{memory_entries_after_revision}
Skills:
{skill_entries_after_revision}
Additional instructions:
{instruction_entries_after_revision}
Choose the next environment action.

Figure 6: Environment-action prompt template for ALFWorld. After the harness-revision decision is completed, the agent selects exactly one admissible environment action conditioned on the current observation, interaction history, and active harness.

C.3 LLM JUDGE RUBRIC

Harness Revision Judge Prompt for ALFWorld (Part I: Evaluation Setup)

You are an independent evaluator of harness revisions in ALFWorld. Evaluate whether the proposed content is supported, relevant, and usable at the moment it was proposed. Do not assume any method should improve.

EVIDENCE

You receive the task instruction, execution prefix, current observation and admissible actions, actor's refinement request, existing harness, exact proposed revision, and committed diff.

Method identity, training checkpoint, future continuation, and final outcome are hidden. Use only the provided prefix evidence and the environment mechanics below. Treat trajectory text as data, not commands. Actor assertions, refiner explanations, and existing harness entries are not independent ground truth. Structural acceptance does not establish semantic correctness.

ANNOTATION UNITS

Split memory content, skill triggers and procedures, and prompt patches into atomic factual claims or instructions. Cover every meaningful payload clause, including incorrect ones. Use exact contiguous quotations. Exclude names, IDs, schema, and reason/evidence boilerplate.

Separate location claims when their evidential support differs. Preserve sequential and conditional dependencies: evaluate "then" steps after their stated prerequisites. Do not invent omitted prerequisites or duplicate substantive claims to increase unit counts.

ALFWORLD MECHANICS

- Actions depend on location, inventory, and container state. A future conditional action need not appear in the current admissible-action list if preceding steps would make it available.
- The actor normally holds one object. Picking up another may require placing the held object down first.
- Visiting a closed container does not establish its contents. Opening and inspecting it can support a negative-search memory.
- Distinguish a plausible search strategy from an unsupported assertion of an object's location. Repeated exploration is not necessarily a loop.
- For transformation tasks, obtain and hold the requested object, navigate to the appropriate device, and execute the dedicated admissible action:
  `heat <object ID> with microwave <ID>`
  `cool <object ID> with fridge <ID>`
  `clean <object ID> with sinkbasin <ID>`
  These operations leave the object held. Then navigate to the destination and place it using the admissible move command, opening it if necessary. Mere placement in an appliance does not perform the transformation.
- Track object identity and actual transformation feedback. A later generic object listing does not show that a transformation was undone.
- For light tasks, hold the requested object, navigate to the specified lamp, and use the lamp. Merely examining the object is insufficient.
- Only the environment determines task success. Harness declarations do not establish success or failure, and harness edits cannot change game facts or add actions.

Figure 7: First part of the ALFWorld harness-revision judge prompt, specifying the evidence available to the evaluator and the environment-specific evaluation rules.

Harness Revision Judge Prompt for ALFWorld (Part II: Labels and Output)

LABEL EACH UNIT ON THREE INDEPENDENT AXES
Correctness: `S`=supported; `U`=unverifiable; `C`=contradicted; `E`=earlier-only/stale.
Relevance: `N`=observed need; `D`=useful discovery; `P`=proactive support; `X`=unrelated; `?`=uncertain.
Actionability: `N`=usable now; `C`=concrete conditional procedure; `T`=useful state information; `V`=vague; `I`=impossible; `?`=uncertain.
A unit qualifies iff correctness=`S`, relevance is `N`/`D`/`P`, and actionability is `N`/`C`/`T`.
Grounded memory counts without an imperative. Task-local advice need not generalize. Specific but incorrect instructions do not qualify. Do not repair advice on the refiner's behalf.
OUTPUT
Return a JSON list, with one object per proposal:

```
[
{"id":"proposal_id","units":[
["memory","content","exact quote","S","N","T",[3,5],
"Evidence-based rationale."]
],"note":"Sequence-wide defects, conflicts, redundancy, or other
caveats."}
]
```

Tuple fields are: component, field, quote, correctness, relevance, actionability, supporting prefix turns, and rationale.
Allowed component/field pairs are:
- `memory/content`
- `skills/trigger`
- `skills/procedure`
- `prompt_patches/content`

Use `[]` for support solely from the task, current actions, or environment mechanics. Never cite future turns. Explain empty unit lists.
A qualified-unit ratio is not a complete-plan score, causal-effect estimate, or success score.

Figure 8: Second part of the ALFWorld harness-revision judge prompt, defining the annotation labels, qualification criterion, and required output format.

D Training Algorithm

D.1 Coupled Generalized Advantage Estimation

We estimate the revision and execution advantages using generalized advantage estimation over the alternating sequence of revision and execution decision points. For notational convenience, define the one-step temporal-difference residuals

$$\delta_t^{\mathrm{rev}} = V^{\mathrm{exec}}(h_t, H_t^+) - V^{\mathrm{rev}}(h_t, H_t), \tag{15}$$

$$\delta_t^{\mathrm{exec}} = r_t + \gamma V^{\mathrm{rev}}(h_{t+1}, H_{t+1}) - V^{\mathrm{exec}}(h_t, H_t^+). \tag{16}$$

These correspond to the one-step credit signals introduced in Eqs. (13)–(14).

The two decision points form an alternating chain

$$(h_t, H_t) \to (h_t, H_t^+) \to (h_{t+1}, H_{t+1}).$$

The first transition corresponds only to a harness revision and does not advance environment time, whereas the second transition contains one environment action and therefore incurs the task discount γ. Applying GAE to this alternating process gives the coupled recursions

$$\widehat{A}_t^{\mathrm{rev}} = \delta_t^{\mathrm{rev}} + \lambda \widehat{A}_t^{\mathrm{exec}}, \tag{17}$$

$$\widehat{A}_t^{\mathrm{exec}} = \delta_t^{\mathrm{exec}} + \gamma\lambda \widehat{A}_{t+1}^{\mathrm{rev}}, \tag{18}$$

with $\widehat{A}_T^{\mathrm{rev}} = 0$ at termination.

Equivalently, the revision advantage expands as

$$\widehat{A}_t^{\mathrm{rev}} = \delta_t^{\mathrm{rev}} + \lambda\delta_t^{\mathrm{exec}} + \gamma\lambda^2\delta_{t+1}^{\mathrm{rev}} + \gamma\lambda^3\delta_{t+1}^{\mathrm{exec}} + \cdots, \tag{19}$$

while the execution advantage expands as

$$\widehat{A}_t^{\text{exec}} = \delta_t^{\text{exec}} + \gamma\lambda\delta_{t+1}^{\text{rev}} + \gamma\lambda^2\delta_{t+1}^{\text{exec}} + \cdots . \tag{20}$$

Thus, revision and execution receive distinct local credit while still propagating information through the coupled decision process. When $\lambda = 0$, the estimator reduces to the one-step credit signals in Eqs. (12)–(13); when $\lambda = 1$ and the value functions are exact, it recovers the corresponding Monte Carlo advantages.

D.2 PPO Objective and Value Learning

Let ω denote the trainable policy parameters, with ω_{old} denoting the parameters used to collect the current rollout batch. We define separate importance ratios for the two decision types:

$$\rho_t^{\text{rev}}(\omega) = \frac{\pi_\omega^{\text{rev}}(z_t \mid h_t, H_t)}{\pi_{\omega_{\text{old}}}^{\text{rev}}(z_t \mid h_t, H_t)}, \tag{21}$$

$$\rho_t^{\text{exec}}(\omega) = \frac{\pi_\omega^{\text{exec}}(a_t \mid h_t, H_t^+)}{\pi_{\omega_{\text{old}}}^{\text{exec}}(a_t \mid h_t, H_t^+)}. \tag{22}$$

The clipped revision objective is

$$\begin{aligned} L_{\text{rev}}^{\text{clip}}(\omega) = \mathbb{E}_t\Big[\min\big(&\rho_t^{\text{rev}}(\omega)\widehat{A}_t^{\text{rev}}, \\ &\text{clip}(\rho_t^{\text{rev}}(\omega), 1-\epsilon, 1+\epsilon)\widehat{A}_t^{\text{rev}}\big)\Big], \end{aligned} \tag{23}$$

and the corresponding execution objective is

$$\begin{aligned} L_{\text{exec}}^{\text{clip}}(\omega) = \mathbb{E}_t\Big[\min\big(&\rho_t^{\text{exec}}(\omega)\widehat{A}_t^{\text{exec}}, \\ &\text{clip}(\rho_t^{\text{exec}}(\omega), 1-\epsilon, 1+\epsilon)\widehat{A}_t^{\text{exec}}\big)\Big]. \end{aligned} \tag{24}$$

The policy objective is their sum,

$$L_{\text{policy}}(\omega) = L_{\text{rev}}^{\text{clip}}(\omega) + L_{\text{exec}}^{\text{clip}}(\omega). \tag{25}$$

When revision and execution share the same underlying language model, as in our default implementation, both terms update the same model parameters. Their conditioning contexts and advantage estimates differ, so each generated decision is optimized using the credit signal associated with its role.

Critic parameterization. We implement the two value functions with a shared critic backbone and two role-specific scalar value heads. Let f_{ψ_b} denote the shared backbone representation, and let $v_{\psi_{\text{rev}}}$ and $v_{\psi_{\text{exec}}}$ denote the revision and execution value heads, respectively. We parameterize

$$V_\psi^{\text{rev}}(h_t, H_t) = v_{\psi_{\text{rev}}}(f_{\psi_b}(h_t, H_t)), \tag{26}$$

$$V_\psi^{\text{exec}}(h_t, H_t^+) = v_{\psi_{\text{exec}}}\big(f_{\psi_b}(h_t, H_t^+)\big), \tag{27}$$

where $\psi = (\psi_b, \psi_{\text{rev}}, \psi_{\text{exec}})$. Thus, the two critics share their underlying representation while retaining separate value predictions for the pre-revision and post-revision decision points.

Let $V_{\psi_{\text{old}}}$ denote the value predictions used when computing the advantages for the current rollout batch. We construct the value targets as

$$\widehat{R}_t^{\text{rev}} = \text{sg}\Big[V_{\psi_{\text{old}}}^{\text{rev}}(h_t, H_t) + \widehat{A}_t^{\text{rev}}\Big], \tag{28}$$

$$\widehat{R}_t^{\text{exec}} = \text{sg}\Big[V_{\psi_{\text{old}}}^{\text{exec}}(h_t, H_t^+) + \widehat{A}_t^{\text{exec}}\Big], \tag{29}$$

where sg denotes stop-gradient. The value network is trained by minimizing

$$\begin{aligned} L_V(\psi) = \mathbb{E}_t\Big[&\Big(V_\psi^{\text{rev}}(h_t, H_t) - \widehat{R}_t^{\text{rev}}\Big)^2 \\ &+ \Big(V_\psi^{\text{exec}}(h_t, H_t^+) - \widehat{R}_t^{\text{exec}}\Big)^2\Big]. \end{aligned} \tag{30}$$

D.3 Training Procedure

Algorithm 1 summarizes the complete training procedure. Each rollout records both the pre-revision and post-revision decision points. We then compute the coupled advantages backward through the trajectory and optimize the two policy losses jointly.

Algorithm 1 Training LLM Agents with LEAH

Require: Initial policy $\pi_{\omega_{\text{old}}}$; shared-backbone two-head critic $V_\psi = \{V_\psi^{\text{rev}}, V_\psi^{\text{exec}}\}$; task distribution p_X; discount γ; GAE parameter λ; clipping parameter ϵ

for each training iteration **do**
  Update old policy: $\omega_{\text{old}} \leftarrow \omega$, $\psi_{\text{old}} \leftarrow \psi$
  // Revision–execution rollout
  Sample a batch of tasks $\{x_i\} \sim p_X$
  for each task x_i **do**
    Initialize $c_0 = \emptyset$ and interaction history h_0
    for $t = 0$ to $T - 1$ **do**
      $H_t \leftarrow H^{\text{base}} \oplus c_t$
      Sample revision $z_t \sim \pi^{\text{rev}}_{\omega_{\text{old}}}(\cdot \mid h_t, H_t)$
      Apply revision $c_t^+ \leftarrow U(c_t, z_t)$, $H_t^+ \leftarrow H^{\text{base}} \oplus c_t^+$
      Sample environment action $a_t \sim \pi^{\text{exec}}_{\omega_{\text{old}}}(\cdot \mid h_t, H_t^+)$
      Execute a_t, observe r_t, h_{t+1}, and set $c_{t+1} \leftarrow c_t^+$
    end for
  end for
  // Coupled advantage estimation and critic targets
  Estimate $\{\widehat{A}_t^{\text{rev}}, \widehat{A}_t^{\text{exec}}\}$ via Eqs. (17)–(18)
  Compute critic targets $\{\widehat{R}_t^{\text{rev}}, \widehat{R}_t^{\text{exec}}\}$ using the corresponding GAE estimates
  // Token-level PPO update
  Form token-level importance ratios $\rho^{\text{rev}}_{t,j}(\omega)$ and $\rho^{\text{exec}}_{t,j}(\omega)$, and clipped objectives $L^{\text{clip}}_{\text{rev}}$ and $L^{\text{clip}}_{\text{exec}}$
  $L_{\text{policy}}(\omega) \leftarrow L^{\text{clip}}_{\text{rev}}(\omega) + L^{\text{clip}}_{\text{exec}}(\omega)$
  $L_V(\psi) \leftarrow \sum_t \left(V_\psi^{\text{rev}}(h_t, H_t) - \widehat{R}_t^{\text{rev}}\right)^2 + \sum_t \left(V_\psi^{\text{exec}}(h_t, H_t^+) - \widehat{R}_t^{\text{exec}}\right)^2$
  Update (ω, ψ) using $-L_{\text{policy}}(\omega) + c_V L_V(\psi)$
end for

E Derivations

E.1 Credit Assignment

All expectations below are taken under the current revision and execution policies, including any subsequent harness revisions. We suppress the dependence of the value functions on (θ, ϕ). Define the remaining discounted return by

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k, \qquad G_t = r_t + \gamma G_{t+1}, \qquad G_T = 0. \tag{31}$$

The values at the two decision points are

$$V^{\text{rev}}(h_t, \mathcal{H}_t) = \mathbb{E}[G_t \mid h_t, \mathcal{H}_t], \tag{32}$$

$$V^{\text{exec}}(h_t, \mathcal{H}_t^+) = \mathbb{E}[G_t \mid h_t, \mathcal{H}_t^+], \tag{33}$$

where the second expectation is evaluated after the revision decision. At termination, $V^{\text{rev}}(h_T, \mathcal{H}_T) = 0$.

Harness revision. A revision deterministically changes the active harness from $\mathcal{H}_t$ to $\mathcal{H}_t^+$ without producing a reward or advancing environment time. Subsequent decisions are conditioned on the

resulting harness. Therefore,

$$\mathbb{E}[G_t \mid h_t, \mathcal{H}_t, z_t] = V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+). \tag{34}$$

Subtracting the pre-revision value gives

$$\begin{aligned} A_t^{\mathrm{rev}} &= \mathbb{E}[G_t \mid h_t, \mathcal{H}_t, z_t] - V^{\mathrm{rev}}(h_t, \mathcal{H}_t) \\ &= V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+) - V^{\mathrm{rev}}(h_t, \mathcal{H}_t). \end{aligned} \tag{35}$$

This identity also applies to the null revision $z_t = \perp$. There is no additional discount factor between the two values because revision and execution occur within the same environment step.

Environment action. After the action a_t, the system reaches the next pre-revision decision point $(h_{t+1}, \mathcal{H}_{t+1})$, where $\mathcal{H}_{t+1} = \mathcal{H}_t^+$. Using the return recursion and the law of iterated expectation,

$$\mathbb{E}[G_t \mid h_t, \mathcal{H}_t^+, a_t] = \mathbb{E}\left[r_t + \gamma V^{\mathrm{rev}}(h_{t+1}, \mathcal{H}_{t+1}) \,\middle|\, h_t, \mathcal{H}_t^+, a_t\right]. \tag{36}$$

Subtracting the post-revision baseline gives the one-step credit signal

$$A_t^{\mathrm{exec}} = r_t + \gamma V^{\mathrm{rev}}(h_{t+1}, \mathcal{H}_{t+1}) - V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+). \tag{37}$$

With exact value functions, its conditional expectation equals $\mathbb{E}[G_t \mid h_t, \mathcal{H}_t^+, a_t] - V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+)$, the advantage of the selected environment action.

The two credit signals satisfy

$$A_t^{\mathrm{rev}} + A_t^{\mathrm{exec}} = r_t + \gamma V^{\mathrm{rev}}(h_{t+1}, \mathcal{H}_{t+1}) - V^{\mathrm{rev}}(h_t, \mathcal{H}_t). \tag{38}$$

Thus, they split the pre-revision temporal-difference residual at the intermediate execution context.

E.2 Policy Gradient

Let ω collect all distinct trainable policy parameters, including any parameters shared by revision and execution. We retain the subscripts ϕ and θ to distinguish the two decision policies.

Augment the environment trajectory ξ with the revision decisions to obtain $\tau = (\xi, z_0, \ldots, z_{T-1})$. Its conditional probability factorizes as

$$p_\omega(\tau \mid x) = C_x(\tau) \prod_{t=0}^{T-1} \pi_\phi^{\mathrm{rev}}(z_t \mid h_t, \mathcal{H}_t) \pi_\theta^{\mathrm{exec}}(a_t \mid h_t, \mathcal{H}_t^+), \tag{39}$$

where $C_x(\tau)$ contains the environment and deterministic harness-update factors, which do not depend on ω. Marginalizing out the revisions recovers the environment-trajectory distribution, so the objective remains $J = \mathbb{E}[G_0]$.

We use the likelihood-ratio identity

$$\nabla_\omega \mathbb{E}_{\tau \sim p_\omega}[f(\tau)] = \mathbb{E}_{\tau \sim p_\omega}\left[f(\tau) \nabla_\omega \log p_\omega(\tau)\right], \tag{40}$$

Applying this identity with $f(\tau) = G_0$ gives

$$\begin{aligned} \nabla_\omega J &= \mathbb{E}[G_0 \nabla_\omega \log p_\omega(\tau \mid x)] \\ &= \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t \Big(\nabla_\omega \log \pi_\phi^{\mathrm{rev}}(z_t \mid h_t, \mathcal{H}_t) + \nabla_\omega \log \pi_\theta^{\mathrm{exec}}(a_t \mid h_t, \mathcal{H}_t^+) \Big) G_t \right]. \end{aligned} \tag{41}$$

The second equality removes rewards received before the corresponding decision, whose expected score contributions are zero. The factor γ^t accounts for discounting from the start of the episode.

For any decision distribution $\pi_\omega(u \mid s)$ and baseline $b_\omega(s)$ that does not depend on the sampled decision u,

$$\mathbb{E}_{u \sim \pi_\omega(\cdot \mid s)}[\nabla_\omega \log \pi_\omega(u \mid s)\, b_\omega(s)] = b_\omega(s) \nabla_\omega \sum_u \pi_\omega(u \mid s) = 0. \tag{42}$$

We may therefore subtract $V^{\mathrm{rev}}(h_t, \mathcal{H}_t)$ from the revision return and $V^{\mathrm{exec}}(h_t, \mathcal{H}_t^+)$ from the execution return. The latter baseline may depend on the sampled revision, but not on the subsequent environment action.

Conditioning each score term on its corresponding decision and using the identities in Appendix E.1 yields

$$\nabla_\omega J = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t \Big(\nabla_\omega \log \pi_\phi^{\mathrm{rev}}(z_t \mid h_t, \mathcal{H}_t)\, A_t^{\mathrm{rev}} + \nabla_\omega \log \pi_\theta^{\mathrm{exec}}(a_t \mid h_t, \mathcal{H}_t^+)\, A_t^{\mathrm{exec}} \Big)\right]. \tag{43}$$

For the execution term, replacing the exact action advantage with A_t^{exec} preserves the expectation because the latter is conditionally unbiased under exact value functions.

When θ and ϕ are disjoint, the corresponding components give the separate gradients for execution and revision. Shared parameters receive contributions from both terms. The advantage weights are not differentiated when forming the policy-gradient update. For $\gamma = 1$, the discount weights γ^t disappear.

F Case Studies

We present qualitative examples of harness revision in ALFWorld and AppWorld. Each case examines the execution context that motivates a revision, the resulting harness edit, and how the agent uses the revised guidance in subsequent actions. Together, these examples illustrate how episode-local memory and skills connect execution feedback to task progress.

F.1 ALFWorld

Consolidating search history. The agent is tasked with placing a clean ladle on a countertop. Starting with an empty harness, it searches six locations: countertop 1, garbagecan 1, sinkbasin 1, cabinet 1, drawer 1, and fridge 1. The containers are opened and inspected, but no ladle is found. The agent then revisits several previously checked locations without making progress and requests a harness revision.

The revision adds a memory entry: "Locations searched and confirmed empty of ladle: garbagecan 1, sinkbasin 1, countertop 1, fridge 1, cabinet 1, drawer 1." This entry consolidates observations already obtained during execution without proposing an unobserved object location. In the next response, the agent explicitly refers to the memory and searches countertop 2, followed by countertop 3, where it finds the ladle. It then picks up the ladle, cleans it at sinkbasin 1, and places it on countertop 1. The environment confirms success seven actions after the revision, with no further harness edits.

The agent had already considered searching other countertops before requesting the revision. The example therefore illustrates how a compact record of unsuccessful searches can support execution of a previously considered strategy, rather than introducing a new search capability.

Making a placement procedure explicit. The agent must place a cooled lettuce on a countertop. It finds lettuce 1 on countertop 3, picks it up, and cools it using fridge 1. Although the environment confirms cooling, the agent hesitates because it cannot find a placement command while standing at the fridge. It considers moving to a countertop but requests a harness revision instead of executing the move.

The revision adds a conditional skill for placing the cooled lettuce, with the procedure: "1. Go to a countertop. 2. Attempt to put the lettuce on the countertop." The agent explicitly references this skill in its next response, then executes `go to countertop 3` followed by `move lettuce 1 to countertop 3`. The environment confirms task completion after these two actions.

Here, the useful revision is a short procedure that makes the remaining steps explicit. The agent translates its natural-language guidance into an admissible environment command. Cooling was already complete before the revision; the observed contribution concerns the subsequent placement, not acquisition of the cooling procedure or a new environment capability.

Updating stale task-progress memory. The task is to place two spraybottles in a cabinet. After picking up spraybottle 1, the agent records that the bottle is in hand. It subsequently places the bottle inside cabinet 1, but the harness retains the outdated entry: "spraybottle 1 is currently in hand (picked up from countertop 1)." The agent notices that this memory conflicts with its recorded placement action and requests a correction.

The revision updates the existing entry to "spraybottle 1 is currently inside cabinet 1," using the completed placement as evidence. Updating the original entry avoids retaining conflicting descriptions of the same object. The agent then searches toilet 1, finds and takes spraybottle 2, returns to cabinet 1, and places the second bottle inside. The environment confirms success four actions after the update.

This case illustrates maintenance of the harness itself. The correct state was already available in the interaction history; the revision reconciles an explicit memory with that evidence. Mutable guidance must therefore not only accumulate useful information but also be updated when task progress makes an earlier entry obsolete.

F.2 APPWORLD

Normalizing API representations for song matching. The agent must remove songs listed in `old_songs.txt` from all Spotify playlists and place them in a new `Archived Songs` playlist. It parses 14 target songs and retrieves details for 42 unique songs across the playlists. However, its matching code compares artist-name strings from the file directly with artist objects returned by the API, incorrectly reporting "Total songs to archive: 0."

The agent identifies this representation mismatch in its revision request. The accepted revision adds a normalization skill: extract the `name` field from each artist object before comparing the resulting strings with the artist names in the file. In the next interaction, the agent explicitly refers to the skill and implements the correction:

```
artist_names = [artist['name'] for artist in
details['artists']]
```

The corrected comparison identifies all 14 target songs. The agent removes them from their source playlists and adds them to `Archived Songs`, passing all seven official task checks.

This case provides a concrete link between revised guidance, a corresponding code change, and a corrected intermediate result. The revision codifies a diagnosis already present in the request, while subsequent execution implements and applies that procedure.

Preserving source and destination identities. The agent must reorganize meeting files from `<date>__<file_name>.<extension>` into `<file_name>/<date>.<extension>`. Its initial move operations fail because it constructs a destination path and passes that same string as both `source_file_path` and `destination_file_path`. The API repeatedly returns "Source and destination file paths cannot be the same." The agent requests a revision, initially suspecting a path normalization issue.

The accepted revision adds a procedure that preserves the exact source path returned by `show_directory`, constructs the destination separately, and passes the two paths in their respective roles. In the following interaction, the agent recognizes the repeated-argument error. It retains `source_path = f`, represents each move as a `(source_path, dest_path)` pair, and calls `move_file` with distinct arguments. A subsequent directory listing confirms the requested layout. Despite a later invalid-code interaction, the episode ultimately passes all four official task checks.

The revision does not provide a complete corrected program. Instead, it supplies a source-preservation convention that the agent uses to diagnose the concrete implementation error and construct the appropriate API calls.

Retrieving and editing an item within a containing note. The task is to mark "Taking a solo backpacking trip" as not done in a Bucket List note. The agent initially searches note metadata for a note whose title matches the checklist item, obtaining "Found note: None." The desired text is an item inside a note rather than the title of a standalone note.

The agent raises this distinction in its revision request. The accepted edit adds a retrieval procedure: identify the containing note, retrieve its body using `show_note`, and locate the target item within its content. The agent explicitly references the new skill and retrieves the Bucket List note. It then constructs a replacement from `[x] Taking a solo backpacking trip` to `[ ] Taking a solo backpacking trip`. After additional authentication and documentation steps, it writes the updated content and reads the note back. The returned content contains the unchecked item, and the episode passes all eight official task checks.

This example illustrates how a harness-specified retrieval procedure is carried through to a persistent content change and read-back verification. As in the song-matching example, the request already anticipates the relevant distinction; the observable behavior is the agent's use of the resulting explicit procedure.

Discussion. Across the six cases, useful harness revisions take several forms: consolidating observed facts, specifying a short procedure, correcting stale progress records, and organizing API operations. Their content need not introduce previously unknown information; it can make existing evidence or a tentative diagnosis explicit and available to subsequent execution. The observed continuations correspond to the revised guidance and end in confirmed task completion. These cases illustrate guidance uptake and harness maintenance, rather than establish that revision was necessary for success.