EVOHARNESS-RL: LEARNING RUNTIME HARNESS COORDINATION FOR SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents increasingly rely on external execution support to maintain state, track progress, recover from failures, and reuse experience across extended interactions. Yet existing harnesses and their use are often tailored to environments and controlled through prompts, heuristics, or system-specific rules, making agent and harness coordination difficult to jointly optimize. We introduce **EVOHARNESS-RL**, a unified framework that separates environment-specific harness implementations from a shared policy-facing interface. EVOHARNESS-RL organizes external support into a Belief, Progress, and Experience (BPE) workspace and exposes four compact harness actions for accessing and updating this state. We first instantiate BPE as an inference-time scaffold and then make harness coordination learnable through supervised initialization followed by cost-aware GRPO. Across heterogeneous long-horizon tasks, EVOHARNESS-BASE improves the average success rate of frontier models by 10.0 percentage points, while EVOHARNESS-RL outperforms the strongest open-source baseline by 8.5 percentage points, together with higher RL rollout efficiency and stronger generalization to unseen tasks. Our analyses show that training gradually shifts agents from frequent scaffold use toward selective, environment-dependent harness access as the policy becomes more capable, while the external workspace continues to evolve and refine itself to better support task execution and generalization. Together, these results show that long-horizon agents benefit not only from external scaffolding itself, but also from learning how and when to coordinate with external support as part of a cost-aware policy.

1 INTRODUCTION

LLM-based agents are increasingly deployed in long-horizon interactive settings, where reliable task execution requires reasoning and acting over extended interactions. In embodied interaction, web navigation, software engineering, and workflow automation (Shridhar et al., 2021; Hong et al., 2026; Yang et al., 2024; Zhou et al., 2024), agents need to maintain environment state, track task progress, recover from failed actions, and reuse knowledge from prior experience. These demands have motivated a growing ecosystem of external execution support, which we broadly refer to as the agent *harness*. Such support can take many forms, ranging from memories and state trackers to tools, verifiers, execution logs, and other mechanisms that augment the agent during interaction (Suzgun et al., 2026; Schick et al., 2023; Ning et al., 2026; Wei et al., 2025; Jiang et al., 2026).

Despite this common role, harnesses are typically tailored to individual environments, leading to increasingly specialized and complex designs that are difficult to transfer across environments or agent models (Young, 2025; Lee et al., 2026; Lou et al., 2026; Jiang et al., 2026; Chen et al., 2026). For example, embodied agents rely on world-state tracking, while web agents rely on page-state tracking and element grounding (Shridhar et al., 2021; Zhou et al., 2024). Beyond what information is maintained, how the agent interacts with the harness is also largely manually designed, often through prompts, heuristics, or system-specific rules rather than jointly learned with the agent. Yet even with the same harness, different models and tasks need different levels of harness support, and these needs change as the policy improves. Learning when and how to use the harness can therefore improve task execution while avoiding unnecessary interaction costs. However, the current fragmented harness designs and interaction rules make such coordination difficult to formulate as a common learning problem. This raises a central question: *How can heterogeneous, task-specific harness designs and interaction rules be unified into a learnable policy for long-horizon execution?*

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.

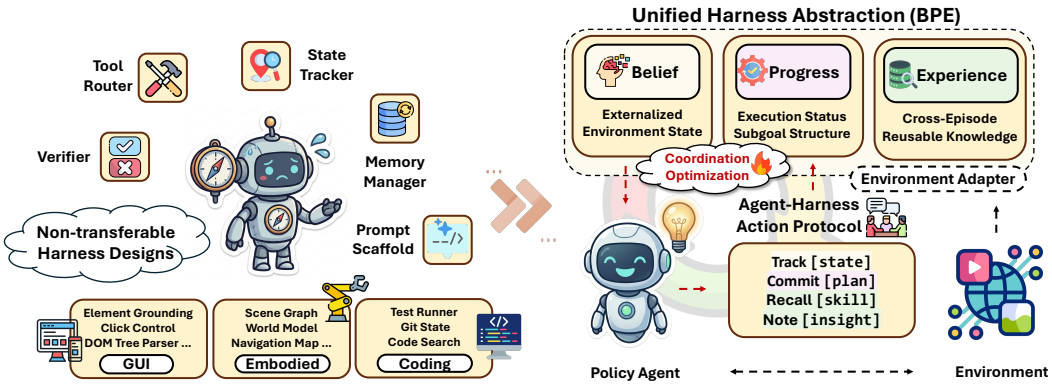

Figure 1: Overview of EVOHARNESS-RL. Long-horizon agents rely on heterogeneous external execution support that is often manually designed and controlled. EVOHARNESS-RL organizes this support into a shared BPE workspace: *Belief* for environment state, *Progress* for execution progress, and *Experience* for reusable knowledge. The agent coordinates with this workspace through four compact actions, track, commit, recall, and note, and learns when these operations are useful during task execution.

To address this challenge, we propose **EVOHARNESS-RL**, a unified framework for learning coordination between an agent and its evolving harness through reinforcement learning. EVOHARNESS-RL separates environment-specific harness implementations from a shared policy-facing interface, organizing external support into a BPE workspace with three roles: *Belief* for environment state, *Progress* for execution progress and subgoals, and *Experience* for reusable knowledge across episodes (Singh et al., 2026; Wang et al., 2026; Shinn et al., 2023). We first expose this workspace as an inference-time scaffold in EVOHARNESS-BASE, and then make harness coordination learnable through four compact actions: track, commit, recall, and note. We bootstrap this interface from successful trajectories and further optimize harness use with cost-aware GRPO (Shao et al., 2024). A transient diversity reward encourages exploration of harness operations early in training, while a success-gated efficiency reward promotes increasingly selective coordination.

We evaluate EVOHARNESS-RL across multiple model backbones and heterogeneous long-horizon agent tasks (Shridhar et al., 2021; Yao et al., 2022a; Zhou et al., 2024). The shared BPE scaffold improves average success rate by 10.0 percentage points in inference time execution across multiple frontier models, while learned harness coordination improves average success rate by 8.5 percentage points over the strongest open-source baseline. By externalizing state, progress, and reusable experience, the scaffold supports early exploration, improves RL sample efficiency, and reduces reliance on environment-specific behaviors, leading to stronger generalization on unseen tasks, with a cross-environment transfer experiment further suggesting the potential for faster adaptation to a new environment. Beyond aggregate performance, our empirical analyses reveal three characteristic dynamics. *Harness annealing:* training progressively internalizes recurring harness-use patterns into the policy, shifting from frequent scaffold-like calls toward selective external state access. *Coordination specialization:* different environments preserve different harness operations, suggesting that training learns an environment-dependent boundary between what should be internalized by the policy and what should remain externalized. *Harness evolution:* online progress updates keep the workspace aligned with ongoing execution, while cross-episode experience consolidation refines knowledge to support generalization to unseen tasks.

Our contributions are three-fold:

- **Unified Harness Abstraction.** We introduce BPE as a shared policy-facing abstraction for heterogeneous harness designs, separating environment-specific implementations from agent-side coordination. The same abstraction supports both inference-time scaffolding with EVOHARNESS-BASE and trainable harness coordination with EVOHARNESS-RL.
- **Learnable Coordination.** We formulate harness use as a learnable coordination problem through a compact agent-harness action protocol, supervised initialization, and cost-aware reinforcement learning. Our theoretically grounded learning paradigm guides the policy from broad exploration of harness operations toward selective, task-adaptive specialization.

- **Empirical Insights.** Across three heterogeneous agent environments and multiple model families, our framework delivers substantial gains in task success through inference-time scaffolding and learned harness coordination. Experiments also show that EVOHARNESS-RL improves RL rollout efficiency and strengthens generalization to unseen tasks. We also identify agent and harness co-evolution dynamics, including harness annealing, coordination specialization, and harness evolution, which further show how training shapes the policy to coordinate more effectively with its external workspace.

2 METHOD

We introduce EVOHARNESS-RL, a trainable framework for coordinating an agent with its external harness. Section 2.1 defines a unified BPE abstraction and its environment adapters; Section 2.2 presents a compact protocol for accessing and updating external state; and Section 2.3 describes the two-stage coordination training recipe.

2.1 UNIFIED HARNESS ABSTRACTION: BELIEF, PROGRESS, AND EXPERIENCE

Long-horizon agents interact over extended contexts and often operate in partially observable environments. As trajectories grow, they may lose track of environment state, forget task progress, and repeat mistakes that could have been avoided using prior experience (Shinn et al., 2023; Wang et al., 2026; Singh et al., 2026). Existing systems often address these failures through domain-specific harness components, such as state trackers, task planners, or episodic memory banks. However, these components typically expose heterogeneous interfaces and rely on manually designed invocation rules, making them difficult to transfer across domains or optimize jointly with the agent policy.

We therefore abstract and unify the policy-facing harness state into three complementary functional roles: **Belief**, **Progress**, and **Experience** (BPE), corresponding to what the agent believes, how the task is progressing, and what can be reused from prior experience. BPE defines shared functional roles rather than a fixed implementation. An **environment adapter** grounds these roles and their corresponding harness operations in each domain while preserving the same policy-facing interface. Details of the BPE instantiations, including their corresponding low-level tools and state representations, are provided in Appendix I. Formally, at each step t, the harness maintains:

$$\mathcal{H}_t = (B_t, P_t, E_t). \tag{1}$$

Belief (B_t). Belief maintains persistent within-episode state derived from agent-visible interaction. Its grounding depends on the environment: embodied agents may use a world-state tracker over objects and locations, shopping agents maintain product and constraint state, and general web agents preserve application state across page transitions. Thus, Belief provides a shared persistent-state abstraction over different domain-specific representations.

Progress (P_t). Progress maintains task decomposition and execution status, recording what has been completed and what remains unresolved. Depending on the domain, this may represent manipulation subgoals, unmet product requirements, or pending stages of a multi-step workflow.

Experience (E_t). Experience maintains reusable cross-episode knowledge, organized into *general experience* shared across tasks and *task-specific experience* specialized to particular task families. We maintain Experience through a *solver-evolver* structure. During rollouts, the solver interacts with the environment using the current Experience and produces new trajectory evidence through execution. The Experience store is kept fixed within each rollout batch. At consolidation epochs, the evolver processes the accumulated trajectories, outcomes, and notes, abstracts reusable general and task-specific experience, and reconciles it with the existing store through ADD, UPDATE, ARCHIVE, and SKIP operations.

We first instantiate BPE as **EVOHARNESS-BASE**, an always-available external scaffold for long-horizon execution. Throughout an episode, the harness continuously maintains and exposes the latest BPE state, while the policy itself selects only environment actions. In this way, task state, execution progress, and reusable experience are explicitly externalized rather than requiring the policy to repeatedly reconstruct or retain them within its own context. This provides persistent

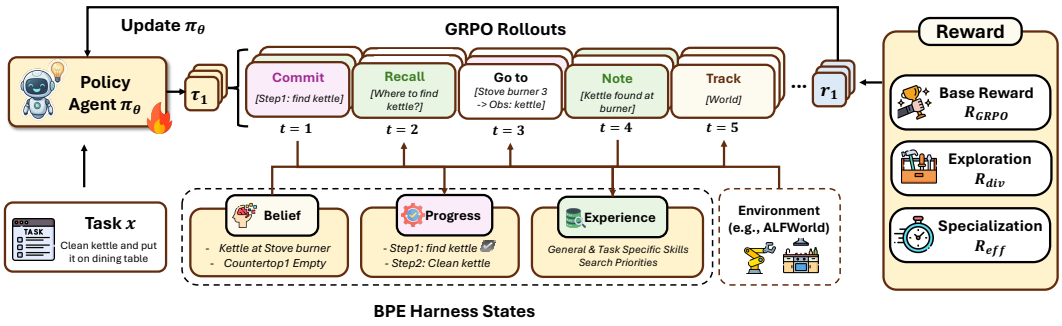

Figure 2: Training pipeline of EVOHARNESS-RL. Given a task, the policy generates rollouts that interleave environment actions with BPE meta-actions, accessing and updating belief, progress, and experience states. GRPO updates the policy using a composite reward that combines the base objective with an annealed exploration bonus and an efficiency incentive, encouraging a transition from broad harness exploration to selective, task-adaptive coordination.

support for long-horizon reasoning without requiring any policy training. However, always-on BPE fixes the boundary between what is maintained externally and what can be handled internally by the policy. As execution progresses and the policy becomes more capable, some external support may become redundant, while other information remains useful to retain outside the model. This motivates learning when to use the harness rather than exposing all BPE support by default.

2.2 AGENT-HARNESS ACTION PROTOCOL

To learn when external support should be used, we expose BPE through a compact set of policy-facing meta actions. Belief maintains environment state, and track retrieves task relevant information from B_t. Progress maintains execution status, and commit records subgoals or progress updates in P_t. Experience maintains reusable cross episode knowledge, where recall retrieves relevant knowledge from E_t and note records new evidence for later consolidation. Together, these operations define the harness action space:

$$\mathcal{A}_{\text{bpe}} = \{\,\text{track}\,,\,\text{commit}\,,\,\text{recall}\,,\,\text{note}\,\}. \tag{2}$$

This protocol preserves distinct BPE roles through a shared interface across environments.

In **EVOHARNESS-RL**, harness access becomes part of the policy decision. Let $\mathcal{A}_{\text{env}}$ denote the environment action space. The policy acts over the joint action space:

$$\mathcal{A} = \mathcal{A}_{\text{env}} \cup \mathcal{A}_{\text{bpe}}. \tag{3}$$

We distinguish the internal harness state $\mathcal{H}_t$ from the harness context currently visible to the policy in EVOHARNESS-RL, denoted by $\mathcal{V}_t$. At step t, the policy receives the environment observation o_t, visible harness context $\mathcal{V}_t$, and task context c_t, and samples:

$$a_t \sim \pi_\theta(\cdot \mid o_t, \mathcal{V}_t, c_t). \tag{4}$$

If $a_t \in \mathcal{A}_{\text{env}}$, the action advances the environment. If $a_t \in \mathcal{A}_{\text{bpe}}$, it accesses or updates the corresponding BPE component and refreshes the visible harness context. Because both types of actions share the same interaction budget, the policy must learn when external support is worth its cost and when the corresponding computation can be handled internally.

2.3 OPTIMIZING AGENT AND HARNESS COORDINATION

Introducing BPE actions expands the policy from environment interaction to joint agent and harness control. This creates a structured optimization problem. The policy must first learn how the new harness actions interact with external state, and then learn when using them is worth the additional interaction cost. We therefore separate these two roles. Supervised learning provides an initial coordination scaffold, while reinforcement learning refines this scaffold into a selective policy.

Supervised initialization. Directly optimizing over $\mathcal{A}_{\text{env}} \cup \mathcal{A}_{\text{bpe}}$ from sparse task feedback is inefficient. Before the policy can learn *when* a harness operation is useful, it must first understand *how* each operation accesses or updates the external BPE state. We therefore use successful teacher trajectories to bootstrap the joint action interface.

The teacher is provided with the same BPE action definitions exposed to the policy and generates trajectories that interleave environment actions with track, commit, recall, and note. We retain successful trajectories and fine-tune the open-source policy on these demonstrations. This stage is not intended to determine the final harness-use strategy. Instead, it initializes a policy that can meaningfully operate in the joint action space, providing a stronger starting point for subsequent RL to explore and refine coordination strategies.

Reinforcement learning for harness coordination. The supervised policy inherits the teacher's harness-use pattern, which may externalize state even when the same decision could be handled internally. The remaining problem is therefore not whether the policy can use the harness, but whether a particular harness interaction is worth its cost in the current trajectory.

Starting GRPO from this initialization changes the role of exploration. Rather than discovering the BPE interface from scratch, the policy can use its rollout budget to compare alternative coordination strategies and learn which external-state operations improve task execution. Meanwhile, this BPE scaffold improves rollout efficiency for success discovery during GRPO training, as analyzed in Appendix F.1. For a rollout trajectory τ collected at RL epoch u, we augment the reward used by standard GRPO with two complementary terms:

$$R(\tau; u) = R_{\text{GRPO}}(\tau) + \underbrace{\lambda_{\text{div}}(u) R_{\text{div}}(\tau)}_{\text{Exploration}} + \underbrace{\lambda_{\text{eff}} R_{\text{eff}}(\tau)}_{\text{Specialization}}. \tag{5}$$

Here, R_{GRPO} denotes the base reward used by our standard GRPO training, including task success and fixed penalties for invalid or degenerate actions. Since environment actions and harness actions consume the same interaction budget, we use $R_{\text{eff}}(\tau) = \mathbf{1}[\text{succ}(\tau)] \frac{T_{\max} - |\tau|}{T_{\max}}$. This rewards successful trajectories that complete the task with fewer total interactions, so unnecessary harness calls reduce the return through the same budget as redundant environment actions.

Annealing exploration into specialization. Efficiency pressure alone can suppress newly introduced BPE actions before the policy has collected enough evidence to discover their utility. We therefore use a transient exploration bonus that encourages coverage of the BPE interface early in training. Specifically, $R_{\text{div}}(\tau) = |\{a \in \mathcal{A}_{\text{bpe}} : a \text{ occurs in } \tau\}| / |\mathcal{A}_{\text{bpe}}|$, which measures the fraction of BPE action types exercised in a trajectory. Its weight follows $\lambda_{\text{div}}(u) = \frac{\lambda_{\text{div}}^{\max}}{2}(1 + \cos(\pi \min(u, U)/U))$, where U is the annealing horizon.

Convergence with diversity annealing. Let J be the final cost-aware objective with maximum J^*, and let $b \sim \mu_u$ be a coordination rule at epoch u, with scheduled-objective residual $\varepsilon_{\text{opt}}(u)$ and diversity weight $\lambda_u = \lambda_{\text{div}}(u)$.

Theorem 1 (Convergence with diversity annealing)**.** *Let $\mathcal{B}^*$ be the set of coordination rules attaining J^*, and let $\Delta_J > 0$ be the reward gap to the best non-optimal rule. Then*

$$J^* - J(\mu_u) \le \varepsilon_{\text{opt}}(u) + \lambda_u, \tag{6}$$

and

$$\Pr(b \in \mathcal{B}^*) \ge 1 - \frac{1}{\Delta_J}\big(\varepsilon_{\text{opt}}(u) + \lambda_u\big). \tag{7}$$

In particular, $\lambda_u = 0$ for all $u \ge U$, so if $\varepsilon_{\text{opt}}(u) \to 0$, then $J(\mu_u) \to J^*$ and $\Pr(b \in \mathcal{B}^*) \to 1$. Full definitions and proofs are provided in Appendix F.2.

Since $R_{\text{div}}(\tau) \in [0, 1]$, the temporary exploration bonus changes the final objective by at most λ_u beyond the current optimization residual. Under our cosine schedule, $\lambda_u = 0$ after annealing, so the diversity term introduces no further objective-level distortion and retained harness use is determined only by task reward, interaction efficiency, and the remaining optimization error.

Table 1: Main results on ALFWorld, WebShop, and WebArena; **best** marks the column best for each backbone. Qwen3-4B results and three-seed means and standard deviations for EVOHARNESS-RL are in Appendices D and E. ∗ / 🔥 denote frozen/trainable methods; Δ averages the four aggregate-metric differences from paired ReAct. † marks methods whose ALFWorld and WebShop results are transcribed from SkillOS (Ouyang et al., 2026). All values are percentages.

Approach	ALFWorld							WebShop		WebArena					Δ
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	SR	Score	SP	CMS	Maps	G&R	Avg.	
Frontier Models															
ReAct (Claude) ∗	100.0	92.3	100.0	100.0	88.0	95.8	96.4	40.2	57.3	26.7	13.3	30.0	20.0	22.0	–
+ EVOHARNESS-BASE ∗	100.0	**100.0**	100.0	100.0	**96.0**	95.8	**98.6**	**46.4**	**61.5**	**33.3**	**20.0**	**40.0**	**50.0**	**34.0**	**+6.2**
ReAct (GLM) ∗	77.1	69.2	48.1	50.0	20.0	45.8	52.1	46.2	62.8	26.7	13.3	**20.0**	20.0	20.0	–
+ EVOHARNESS-BASE ∗	**94.3**	**84.6**	**74.1**	**68.8**	**52.0**	**87.5**	**77.9**	**49.6**	**70.0**	26.7	**20.0**	10.0	**30.0**	**22.0**	**+9.6**
ReAct (GPT) ∗	74.3	53.8	48.1	62.5	60.0	58.3	60.7	37.8	56.1	26.7	26.7	**10.0**	10.0	20.0	–
+ EVOHARNESS-BASE ∗	**97.1**	**76.9**	**85.2**	**93.8**	**88.0**	**62.5**	**85.0**	**46.2**	**63.9**	**40.0**	**33.3**	0.0	**20.0**	**26.0**	**+11.6**
Open-Source Baselines (Qwen3-8B)															
ReAct ∗	78.1	46.2	33.3	37.5	29.3	47.2	47.8	12.0	42.9	20.0	13.3	0.0	20.0	14.0	–
ExpeL ∗	91.4	76.9	14.8	43.8	28.0	45.8	50.7	7.0	25.7	20.0	26.7	0.0	20.0	18.0	−3.8
ReasoningBank† ∗	83.8	48.7	49.4	39.6	41.3	54.2	56.2	11.4	35.4	20.0	13.3	10.0	10.0	14.0	+0.1
MemP† ∗	80.0	43.6	24.7	33.3	38.7	48.6	47.9	12.0	35.7	20.0	20.0	0.0	10.0	14.0	−1.8
Dynamic Cheatsheet ∗	88.6	53.8	29.6	37.5	20.0	66.7	52.1	5.2	20.4	20.0	**33.3**	0.0	10.0	18.0	−5.3
ACE ∗	85.7	38.5	29.6	37.5	36.0	58.3	51.4	3.2	13.2	20.0	26.7	0.0	10.0	16.0	−8.2
SkillOS-base† ∗	79.0	41.0	45.7	37.5	38.7	55.6	53.1	13.6	38.6	20.0	20.0	0.0	20.0	16.0	+1.2
GRPO 🔥	88.6	69.2	74.1	68.8	48.0	41.7	66.4	73.0	82.4	33.3	13.3	**20.0**	0.0	18.0	+30.8
SkillOS† 🔥	85.7	56.4	54.3	43.8	46.7	62.5	61.2	16.5	40.6	26.7	20.0	10.0	10.0	18.0	+4.9
SkillRL 🔥	94.3	84.6	92.6	87.5	72.0	75.0	85.0	73.2	85.5	20.0	20.0	10.0	20.0	18.0	+36.3
Ours (Qwen3-8B)															
EVOHARNESS-BASE ∗	71.4	53.8	63.0	50.0	48.0	41.7	56.4	18.6	43.7	20.0	26.7	10.0	10.0	18.0	**+5.0**
EVOHARNESS-SFT 🔥	80.0	53.8	88.9	75.0	40.0	62.5	68.6	43.8	48.1	33.3	6.7	0.0	20.0	16.0	**+15.0**
EVOHARNESS-RL 🔥	**100.0**	**92.3**	**96.3**	**100.0**	**84.0**	**95.8**	**95.0**	**80.6**	**90.8**	**40.0**	20.0	10.0	**30.0**	**26.0**	**+43.9**

3 EXPERIMENTS

We evaluate EVOHARNESS-BASE for inference time scaffolding and EVOHARNESS-RL for learned harness coordination across three heterogeneous long horizon agent environments. We compare against strong inference time and trainable baselines, study the contributions of individual components and generalization to unseen tasks, and analyze agent and harness coevolution dynamics.

3.1 EXPERIMENT SETUP

Environments. **ALFWorld** (Shridhar et al., 2021) is a text-based household environment with six task families: pick-and-place (PICK), inspection under light (LOOK), cleaning (CLEAN), heating (HEAT), cooling (COOL), and placing two objects (PICK2). **WebShop** (Yao et al., 2022a) evaluates product search and purchase under compositional constraints. For both benchmarks, we use the SkillRL training and validation splits (Xia et al., 2026). **WebArena** (Zhou et al., 2024) covers realistic multi-site tasks in shopping (SP), content management (CMS), maps, and GitLab/Reddit (G&R). Following AgentGym-RL (Xi et al., 2025), we exclude backend-edit tasks to avoid persistent state changes across on-policy rollouts. Together, these distinct environments test whether our method generalizes across domains and interaction structures in long-horizon agentic tasks.

Baselines. We compare EVOHARNESS-RL against frontier and open-source models. First, we evaluate frontier models from three families: Claude Opus 4.6, GLM-5, and GPT-5. For each model, we report standard ReAct and then apply our inference-time EVOHARNESS-BASE, as described in Section 2.2. Second, we evaluate open-source baselines, all using the same Qwen3-8B backbone. The **frozen inference-time baselines** are ReAct (Yao et al., 2022b), ExpeL (Zhao et al., 2024), ReasoningBank (Ouyang et al., 2025), MemP (Fang et al., 2026), Dynamic Cheatsheet (Suzgun et al., 2026), ACE (Zhang et al., 2025a), and SkillOS-base (Ouyang et al., 2026); these methods use prompting or external memory without updating model parameters. The **training-time baselines** are standard GRPO (Shao et al., 2024), SkillOS (Ouyang et al., 2026), and SkillRL (Xia et al., 2026), which optimize the policy through reinforcement learning or memory-augmented training. We compare them with our variants: EVOHARNESS-BASE, EVOHARNESS-SFT, and EVOHARNESS-RL, instantiated with Qwen3-8B. Detailed baseline descriptions are provided in Appendix J, and implementation details in Appendix K.

Table 2: Ablation results on ALFWorld.

Variant	ALFWorld							Δ
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	
***Training-time BPE ablation for* EVOHARNESS-RL**								
Full BPE	100.0	92.3	96.3	100.0	84.0	95.8	95.0	–
w/o Belief	97.1	76.9	88.9	87.5	84.0	91.7	89.3	▼5.7
w/o Progress	100.0	84.6	100.0	100.0	84.0	70.8	90.7	▼4.3
w/o Experience	97.1	76.9	88.9	81.2	84.0	66.7	84.3	▼10.7
Always-on BPE	97.1	92.3	85.2	100.0	68.0	91.7	88.6	▼6.4
w/o SFT	68.6	53.8	55.6	50.0	52.0	12.5	50.0	▼45.0
SFT + GRPO	100.0	61.5	92.6	81.3	80.0	91.7	87.9	▼7.1
***Reward ablation for* EVOHARNESS-RL**								
Full Reward	100.0	92.3	96.3	100.0	84.0	95.8	95.0	–
w/o Diversity	100.0	100.0	96.3	93.8	84.0	83.3	92.9	▼2.1
w/o Efficiency	100.0	53.8	96.3	75.0	84.0	83.3	86.4	▼8.6

Table 3: Generalization results on the ALFWorld unseen split for ReAct, vanilla SFT + GRPO, and EVOHARNESS-RL variants.

Approach	ALFWorld Unseen SR (%)						
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.
ReAct (Opus 4.6) ∗	100.0	83.3	96.8	78.3	90.5	94.1	91.0
+ EVOHARNESS-BASE ∗	100.0	94.4	100.0	95.7	100.0	94.1	**97.8**
ReAct (Qwen3-8B) ∗	66.7	50.0	48.4	47.8	52.4	29.4	50.0
+ EVOHARNESS-BASE ∗	83.3	38.9	87.1	91.3	90.5	58.8	77.6
+ EVOHARNESS-SFT 🔥	70.8	66.7	80.6	69.6	47.6	76.5	69.4
+ SFT + GRPO 🔥	75.0	66.7	83.9	69.6	90.5	76.5	77.6
+ Always-on BPE 🔥	91.7	83.3	96.8	100.0	81.0	82.4	90.3
+ EVOHARNESS-RL 🔥	95.8	100.0	93.5	91.3	100.0	88.2	**94.8**

3.2 MAIN RESULTS

Table 1 shows that EVOHARNESS-RL consistently benefits both inference time scaffolding and learned harness coordination across heterogeneous agent environments. On frontier models, EVOHARNESS-BASE yields a **10.0 percentage-point** improvement in average success rate over the corresponding ReAct agents, demonstrating that the shared BPE scaffold provides effective long horizon support without parameter updates. In contrast, existing frozen memory and harness baselines on Qwen3-8B show mixed or even negative gains across environments, suggesting that manually designed external support does not necessarily generalize across different agentic settings. Learning the coordination policy further amplifies the benefit: EVOHARNESS-RL yields an **8.5 percentage-point** improvement in average success rate over the strongest trainable baseline. Supervised initialization is less consistent, suggesting that imitation inherits teacher specific harness use patterns, whereas coordination RL learns when and how to use external support.

3.3 ABLATION RESULTS

Table 2 evaluates the contributions of BPE state, the training pipeline, and the reward design. Removing any BPE component reduces average performance, consistent with the inference-time ablations in Appendix G, with *Experience* causing the largest drop due to the recurring procedures and reusable object-location knowledge in ALFWorld. The training ablations further highlight the importance of learned harness coordination. Without SFT initialization, the policy struggles to explore the new BPE action space from sparse rewards. A vanilla SFT + GRPO baseline with the same supervised-data budget also remains below the full method, showing that the gain is not simply due to additional post-training. Beyond final performance, the BPE scaffold also improves rollout efficiency, allowing subsequent GRPO to reach strong validation performance substantially earlier than vanilla SFT + GRPO (Appendix B.2). Similarly, Always-on BPE underperforms EVOHARNESS-RL: without control over when to access or update the harness, the policy receives redundant external context rather than selectively using the most task-relevant information. Together, these results support learning when and how to coordinate with the harness rather than merely exposing it to the agent. Finally, removing either the diversity or efficiency reward lowers performance; Appendix B.3 further analyzes their effects on harness-use dynamics.

3.4 UNSEEN-TASK EVALUATION

Table 3 evaluates whether the benefits of BPE and learned harness coordination persist on unseen tasks. At inference time, EVOHARNESS-BASE continues to improve both frontier and open-source agents without parameter updates, suggesting that the unified BPE scaffold remains useful beyond the seen split. Learned coordination provides a further advantage: EVOHARNESS-RL achieves 94.8% success, corresponding to a 22.2% relative improvement over vanilla SFT + GRPO and closely matching its performance on seen tasks. One possible explanation is a benefit of *cognitive externalization*: by keeping task state, progress, and reusable experience outside the model, BPE reduces the need for the policy to encode environment-specific and task-specific regularities directly in its parameters. This separation may mitigate overfitting to the training instances and preserve more flexible decision making when the agent encounters unseen tasks.

4 Analysis of Agent-Harness Evolution Dynamics

We study how the policy learns selective harness coordination, how this coordination specializes across environments, and how the Experience bank evolves. Together, these dynamics reveal an evolving division of labor between the model and its external harness. Additional results are provided in Appendix B.

4.1 Learning Selective Coordination

We study how harness coordination evolves as the policy improves. Figure 3 shows that harness use declines while task success continues to rise. Early in training, the policy relies more heavily on external support to stabilize long-horizon execution; as recurring interaction patterns are learned, routine decisions are increasingly handled internally, leaving the harness for information that remains difficult or useful to externalize. This shift reflects a transition from broad scaffold dependence to selective externalization. Different BPE actions stabilize at different levels, suggesting that training learns a boundary between what can be internalized by the policy and what remains useful to externalize. Cross-environment results in Appendix G (Figure 5) further show that this boundary varies with the interaction demands of each environment. Finally, a coordination perturbation ablation (Table 7) shows that harness frequency alone does not explain the gain; the learned choice and timing of BPE operations are also important.

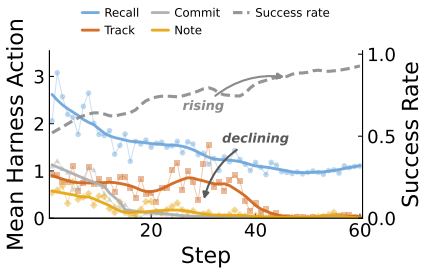

Figure 3: Action-specific harness annealing. Different BPE actions decay at different rates, while the success rate is still rising.

4.2 Learning Environment-Dependent Coordination

We compare harness retention across the three environments. Figure 4 shows that harness use decreases in all three, but the retained meta-actions differ with the task structure. ALFWorld consists of recurring household routines in partially observed scenes, where object locations and manipulation procedures are often reusable across episodes. This structure makes recalled experience especially valuable. WebShop instead presents a new combination of product attributes and purchase constraints in each episode. General shopping procedures transfer, but the policy must primarily preserve the current request and intermediate search state, which favors tracking. WebArena spans heterogeneous websites and longer workflows with changing subgoals, so no single state type is sufficient; its retained actions remain more mixed, with progress management playing a larger role. The key result is therefore not a uniform reduction in harness use. Rather, the same training recipe learns different patterns of harness use depending on the environment. Appendix C further shows that these different usage patterns can still share a transferable underlying coordination strategy: a WebShop-trained EvoHarness-RL adapts to ALFWorld substantially faster than standard GRPO and quickly approaches the in-domain training trajectory. This suggests that the transferable structure lies in the underlying coordination strategy for deciding when external support is useful, rather than in a fixed source-domain pattern of harness actions.

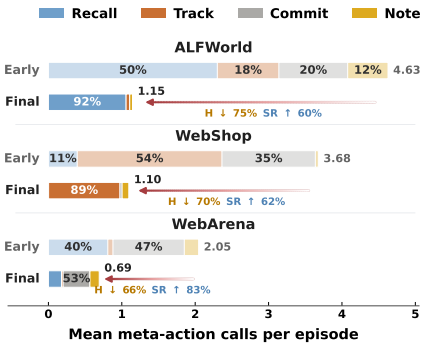

Figure 4: Environment-dependent retention of meta-actions across benchmarks. H denotes the number of harness actions, and SR denotes training success rate.

4.3 Harness Evolution

Table 4 (right) shows a clear transition from acquisition to refinement. Early consolidation cycles are dominated by ADD operations that accumulate general strategies, task-specific procedures, common mistakes, and search priorities. As the bank fills, additions saturate and consolidation shifts toward UPDATE, which incorporates later evidence, corrects stale entries, and reconciles redundant

Setting	Experience Variant	Pick	Look	Clean	Heat	Cool	Pick2	Avg.
Seen	w/o Experience	97.1	76.9	88.9	81.2	84.0	66.7	84.3
	SFT-init bank	100.0	92.3	92.6	100.0	84.0	91.7	93.6
	Self-Consolidate	100.0	100.0	96.3	100.0	84.0	87.5	94.3
	Evolved bank	**100.0**	**92.3**	**96.3**	**100.0**	**84.0**	**95.8**	**95.0**
Unseen	w/o Experience	83.3	61.1	93.5	82.6	95.2	82.4	84.3
	SFT-init bank	70.8	94.4	87.1	87.0	95.2	88.2	86.6
	Self-Consolidate	95.8	94.4	90.3	82.6	81.0	82.4	88.1
	Evolved bank	**95.8**	**100.0**	**93.5**	**91.3**	**100.0**	**88.2**	**94.8**

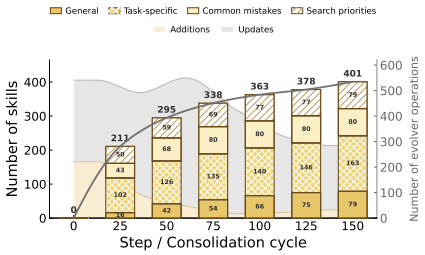

Table 4: Experience evolution during EVOHARNESS-RL training. Left: ALFWorld seen and unseen success rates of separately trained variants under Experience mechanisms. Right: Experience-bank composition and consolidation operations over training; task-specific counts aggregate entries across task-family categories.

or conflicting experience. Experience evolution is therefore not append-only accumulation, but continual maintenance of reusable knowledge as training progresses. Table 4 (left) studies whether Experience should remain fixed or co-evolve with the policy during RL. Each variant is trained separately under its corresponding Experience mechanism, so the comparison reflects how Experience evolution shapes policy learning rather than an inference-time bank swap. A fixed bank provides useful prior knowledge but cannot adapt to the changing errors and behaviors of the policy. Continual consolidation instead forms a feedback loop in which rollout evidence refines Experience and the updated bank guides subsequent policy learning. The larger gain on unseen tasks suggests that this co-evolution improves not only knowledge accumulation but also abstraction and transferability. Self-consolidation with Qwen3-8B remains effective without a frontier consolidator, showing that the pipeline is robust even when Experience evolution is performed by the policy model itself. Appendix H provides qualitative examples of this evolution process.

5 RELATED WORK

We review harness engineering and self-evolving agents; see Appendix A for details.

Harness engineering. LLM agents increasingly rely on external harnesses that structure observations, actions, tools, and execution feedback (Yao et al., 2022b; Schick et al., 2023; Yang et al., 2024; Zhou et al., 2024; Ning et al., 2026). Recent work further optimizes or evolves these systems through state externalization, harness search, trace driven adaptation, and automated harness construction (Jiang et al., 2026; Lee et al., 2026; Chen et al., 2026; Zhang et al., 2026; Shao et al., 2026; Huang et al., 2026; Ke et al., 2026). Existing approaches largely study *what harness should be provided or constructed*. In contrast, EVOHARNESS-RL unifies heterogeneous harness functions behind a shared policy facing interface and learns *how the acting policy coordinates with them at runtime*, including when external support should be accessed, updated, or bypassed.

Memory and self-evolving agents. External memory provides a natural mechanism for context externalization in long horizon agents. Prior work progresses from reusing trajectories and reflections (Shinn et al., 2023; Zhao et al., 2024), to organizing experience into reusable procedural knowledge (Wang et al., 2024; 2025; Ouyang et al., 2025; Fang et al., 2026), and further to continuously evolving external memories and skill repositories (Suzgun et al., 2026; Zhang et al., 2025a; Ouyang et al., 2026; Xia et al., 2026). These methods primarily focus on how memory is represented, retrieved, and evolved within particular settings. EVOHARNESS-RL instead places cross episode Experience together with within episode Belief and Progress under a shared interface, and studies how the policy learns to coordinate with these forms of externalized context during execution.

6 CONCLUSION

We introduced EVOHARNESS-RL, a shared BPE interface that turns coordination with environment-specific harnesses into a learnable decision problem. Across multiple benchmarks and model families, BPE improves inference-time execution, while SFT-initialized, cost-aware reinforcement learning further improves task success, rollout efficiency, and generalization to unseen tasks. Our analyses show that, as the policy improves, harness use becomes more selective and environment-dependent, revealing an evolving division of labor between the policy and external support. Overall, EVOHARNESS-RL provides a step toward jointly optimizing what an agent should internalize and what should remain external during long-horizon execution.

AI USE STATEMENT

During the preparation of this paper, we made limited use of ChatGPT as an auxiliary writing tool for polishing author-written text, including improving grammar, fluency, and readability. The scientific ideas, methodology, experimental design, analysis, and conclusions were developed and validated by the authors. The authors reviewed and take full responsibility for all content in the paper. The use of LLMs was limited to writing assistance and does not constitute authorship.

ETHICS STATEMENT

Our work studies long-horizon LLM agents and their coordination with external harnesses in established research environments. The proposed framework is intended for methodological research on agent learning and execution rather than deployment-specific applications. Like other research on LLM agents, our work may have broader social impacts, but we do not identify any particular ethical concerns that require special attention in this study. All authors have reviewed the ICLR Code of Ethics and affirm adherence to its principles throughout this work.

REPRODUCIBILITY STATEMENT

We will release the full implementation code upon paper acceptance. We provide detailed descriptions of the BPE abstraction, agent and harness action protocol, training procedure, reward design, environment-specific implementations, prompts, and hyperparameters in the main paper and appendix. We additionally report ablations, results across random seeds, training dynamics, and evaluation protocols to facilitate independent reproduction of the reported findings.

REFERENCES

Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22(98):1–76, 2021. URL https://jmlr.org/papers/v22/19-736.html.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.829. URL https://aclanthology.org/2026.findings-acl.829/.

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

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

Yining Hong, Rui Sun, Bingxuan Li, Xingcheng Yao, Maxine Wu, Alexander Chien, Da Yin, Ying Nian Wu, Zhecan Wang, and Kai-Wei Chang. Embodied web agents: Bridging physical-digital realms for integrated agent intelligence. *Advances in Neural Information Processing Systems*, 38, 2026.

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

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.

Zixuan Ke, Vaidehi Patil, Haizhou Shi, Yang Li, Ye Liu, Sarath Shekkizhar, Anurag Koul, Jiayu Wang, Xuan Phi Nguyen, Semih Yavuz, et al. Evoharnessbench: Can your agents keep pace with an evolving harness? *arXiv preprint arXiv:2609.04280*, 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*, 2026.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-r1: Learning to edit executable runtime harnesses from agent failure trajectories. *arXiv preprint arXiv:2608.02276*, 2026.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

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

Joykirat Singh, Zaid Khan, Archiki Prasad, Justin Chih-Yao Chen, Akshay Nambi, Hyunji Lee, Elias Stengel-Eskin, and Mohit Bansal. Agent-brace: Decoupling beliefs from actions in long-horizon tasks via verbalized state uncertainty. *arXiv preprint arXiv:2605.11436*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Taiyi Wang, Sian Gooding, Florian Hartmann, Oriana Riva, and Edward Grefenstette. A subgoal-driven framework for improving long-horizon llm agents. *arXiv preprint arXiv:2603.19685*, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, pp. 63897–63911. PMLR, 2025.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Xinping Lei, Qingshui Gu, Yuxuan Zhang, Zexuan Wang, Chen He, Chen Huang, Maojia Song, et al. Harnessdev: Can llms create and evolve their own agent harness? *arXiv preprint arXiv:2609.01437*, 2026.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, et al. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*, 2025.

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

Lin Xiao. On the convergence rates of policy gradient methods. *Journal of Machine Learning Research*, 23(282):1–36, 2022. URL https://jmlr.org/papers/v23/22-0056.html.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit Bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2111.02080.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

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

Justin Young. Effective harnesses for long-running agents. Anthropic Engineering Blog, November 2025. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. Accessed: 2026-05-11.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 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. *arXiv preprint arXiv:2510.04618*, 2025a.

Yufeng Zhang, Fengzhuo Zhang, Zhuoran Yang, and Zhaoran Wang. What and how does in-context learning learn? Bayesian model averaging, parameterization, and generalization. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 1684–1692. PMLR, 2025b. URL https://proceedings.mlr.press/v258/zhang25d.html.

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

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

APPENDIX CONTENTS

A Related Work 14

Extended discussion of harness engineering and self-evolving agents.

B Training Dynamics 14

Harness annealing and optimization behavior across benchmarks.

C Cross-Domain Transferability 17

Adaptation from WebShop to ALFWorld under different initialization strategies.

D Cross-Backbone Adaptability 18

Results with Qwen3-4B and Qwen3-8B policy backbones.

E Method Robustness Across Random Seeds 18

Three-seed performance on ALFWorld, WebShop, and WebArena.

F Theoretical Analysis 18

Rollout efficiency from BPE information, convergence under diversity annealing, and the scope of both results.

G More Ablation Analysis 23

Harness-schedule, component-removal, and prompt-stripping studies.

H Qualitative Case Study 23

Within-episode coordination and cross-episode Experience evolution.

I Environment-Specific BPE Grounding 25

Concrete Belief, Progress, and Experience representations.

J Baseline Details 29

Implementation notes for the compared agent baselines.

K Implementation Details 30

Models, training settings, horizons, and reward hyperparameters.

L Prompts 32

Runtime harness, message-format, and Experience-consolidation prompts.

A Related Work

Our work is closely related to two lines of research: harness engineering, which studies the external systems that support agent execution, and memory and self-evolving agents, which externalize and adapt task-relevant context across interactions.

Harness engineering. The execution capability of an LLM agent is strongly shaped by its external harness, which determines how observations are represented, how actions and tools are exposed, and what execution feedback is returned to the policy. Early agent paradigms established explicit reasoning and action loops and tool augmented interaction (Yao et al., 2022b; Schick et al., 2023), while subsequent systems introduced increasingly structured execution interfaces, including specialized observation renderers, file system and code access, environment specific action spaces, and execution feedback (Yang et al., 2024; Zhou et al., 2024; Ning et al., 2026). As agent tasks become longer and more complex, recent work has increasingly studied how these surrounding systems can themselves be optimized or evolved. Harness-1 externalizes intermediate search state into environment side memory (Jiang et al., 2026), while Meta-Harness searches over harness code and configurations using execution traces (Lee et al., 2026). HarnessX studies composable and trace driven harness adaptation (Chen et al., 2026), and recent approaches such as Self-harness and Harness-R1 improve executable harnesses from model specific weaknesses or failure trajectories (Zhang et al., 2026; Shao et al., 2026). MemoHarness further studies experience conditioned harness adaptation (Huang et al., 2026), while EvoHarnessBench and HarnessDev study how agents cope with, construct, and evolve changing harnesses (Ke et al., 2026; Wu et al., 2026). These approaches primarily optimize, synthesize, or evaluate *what harness is provided* to an agent. In contrast, EvoHarness-RL abstracts heterogeneous harness mechanisms into a shared policy facing interface and focuses on *runtime agent and harness coordination*, allowing the acting policy to learn when different forms of external support should be accessed, updated, or bypassed jointly with environment execution.

Memory and self-evolving agents. External memory provides a natural form of context externalization for long horizon agents, allowing information that would otherwise need to be retained or reconstructed within the model context to persist across interactions. Early memory augmented agents reuse trajectories, reflections, and induced insights across episodes, as in Reflexion and ExpeL (Shinn et al., 2023; Zhao et al., 2024). Later work increasingly structures such experience into reusable procedural knowledge, including executable skills, workflows, reasoning items, and procedural scripts (Wang et al., 2024; 2025; Ouyang et al., 2025; Fang et al., 2026). More recent systems treat the external context itself as an evolving resource: Dynamic Cheatsheet and ACE continually refine compact memories or structured playbooks (Suzgun et al., 2026; Zhang et al., 2025a), while SkillOS and SkillRL learn to curate or co-evolve external skill repositories together with agent execution (Ouyang et al., 2026; Xia et al., 2026). These methods demonstrate the value of externalizing and evolving context, but primarily focus on how memory is represented, retrieved, updated, or curated within particular settings. In contrast, EvoHarness-RL abstracts heterogeneous forms of externalized context into a shared policy facing interface, organizing within episode Belief and Progress together with cross episode Experience while leaving their concrete implementations environment specific. Our focus is on learning how the acting policy coordinates with these external states at runtime, and on analyzing how this coordination changes during training as different forms of support are increasingly handled internally or remain useful to access externally.

B Training Dynamics

We provide additional analyses of learned agent–harness coordination by comparing training-success dynamics and harness use across environments.

B.1 Environment-Dependent Harness Dynamics

Figure 5 shows that harness use decreases across all three environments, but different operations are retained because the environments impose different information demands. In ALFWorld, tasks repeatedly reuse household procedures and object-location priors across episodes. This makes recalled Experience persist throughout training, while explicit Progress and newly written notes become less necessary once recurring manipulation patterns are absorbed by the policy. WebShop has a different

structure: each episode introduces a new combination of product attributes and purchase constraints, so cross-episode retrieval is less directly useful, whereas maintaining the current request, candidate attributes, and verification status remains important. Consequently, Track is retained more strongly as an episode-specific state mechanism. WebArena is both more heterogeneous and more workflow dependent. Tasks span different websites and multi-stage interactions whose relevant state and subgoals change across pages, producing a more mixed pattern rather than a single dominant harness operation. Recall is useful early for reusable interaction routines but declines substantially as common navigation behaviors are learned, while state- and progress-related operations remain useful for coordinating instance-specific workflows.

These differences suggest that annealing is not simply uniform sparsification of harness calls. Instead, training appears to preserve external access when information remains difficult to encode parametrically or varies substantially across episodes, while recurring and predictable coordination patterns are increasingly handled by the policy itself. The resulting retention pattern therefore reflects the interaction between task regularity, cross-episode reusability, and within-episode state dependence.

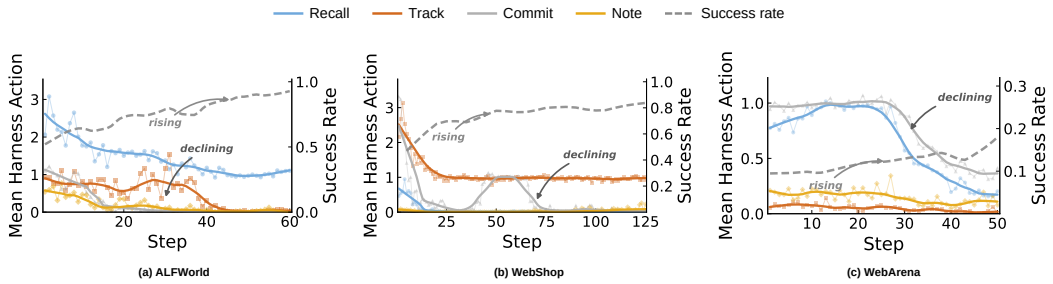

Figure 5: Environment-dependent harness dynamics across benchmarks.

B.2 Training and Rollout Efficiency

Figure 6 compares GRPO with the full EvoHarness-RL training pipeline on ALFWorld. EvoHarness-RL enters the high-success regime substantially earlier, with its smoothed success rate first reaching 80% at step 19, while GRPO improves more slowly and remains more variable. Over the final ten training steps, EvoHarness-RL achieves an average success rate approximately 23 percentage points higher than GRPO. The earlier rise and stronger tail performance indicate that the harness-aware pipeline extracts useful learning signal more efficiently from the rollout budget.

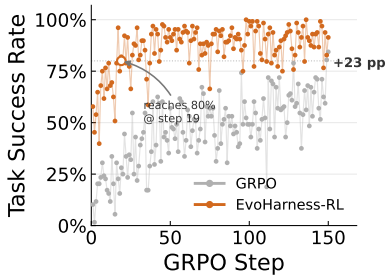

Figure 6: ALFWorld training-success dynamics for GRPO and EvoHarness-RL.

To separate this gain from SFT initialization alone, Figure 7 compares against SFT + GRPO under the same supervised-data budget. The figure reports performance on the validation set during training, evaluated every five GRPO steps. At the first evaluation, SFT + GRPO has the higher validation success rate (79.7% versus 57.8%). This initial gap reflects a harder supervised-learning problem for EvoHarness-RL: with the same number of examples, SFT + GRPO only needs to imitate environment-action patterns, whereas our policy must jointly learn those task-solving patterns, the semantics and syntax of four additional harness actions, and when each action is worth invoking. The resulting policy therefore begins RL with less capacity devoted to direct task execution.

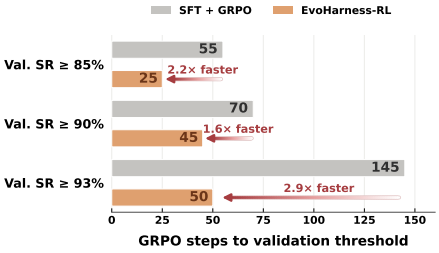

Figure 7: GRPO steps to reach ALFWorld validation-set success thresholds, evaluated every five training steps.

The ordering reverses as optimization proceeds. Once the harness-action interface is established, the BPE scaffold gives GRPO structured access to belief, progress, and reusable experience. This

reduces repeated state reconstruction and unproductive exploration, while making successful coordination patterns easier to reinforce. Consequently, EVOHARNESS-RL reaches 85%, 90%, and 93% validation success with 2.2×, 1.6×, and 2.9× speedups in GRPO steps, respectively. The lower initial success but faster subsequent improvement shows that the gain is not a consequence of a stronger SFT checkpoint alone; rather, the learned scaffold makes downstream GRPO substantially more sample efficient.

Proposition F.1 provides a conditional mechanism for this empirical trend: useful BPE information can reduce the rollout budget needed to discover a successful trajectory at a fixed policy checkpoint. The training curves additionally capture policy updates and experience-bank evolution.

B.3 REWARD-SHAPING DYNAMICS

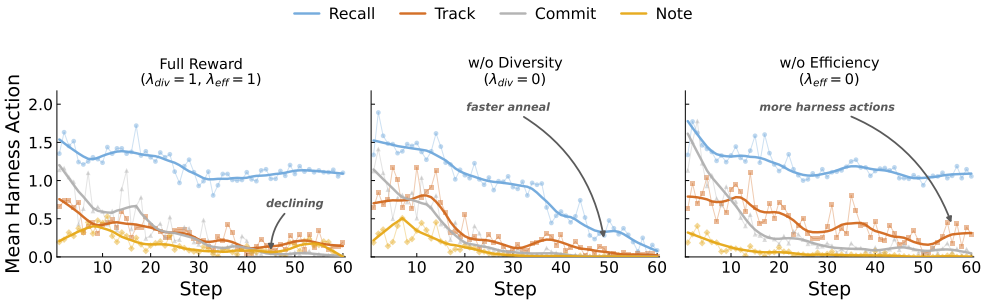

Figure 8: Harness-action dynamics under different reward designs. Without diversity, harness use anneals more quickly; without efficiency, it remains higher later in training.

Figure 8 explains how the two auxiliary rewards produce the performance differences in Table 2. Without the diversity reward, aggregate BPE use falls earlier and more rapidly. We interpret this decline as premature pruning rather than successful specialization: the policy receives less incentive to try newly introduced harness operations before it has gathered enough evidence about their utility. The diversity term therefore acts as a temporary exploration mechanism that keeps multiple coordination strategies available early in training.

Without the efficiency reward, harness use remains elevated later in training. This indicates that task reward alone can preserve useful external operations, but provides little pressure to distinguish necessary calls from redundant ones. The efficiency term supplies this pressure by rewarding successful trajectories that solve the task with fewer total interactions. Together, the two rewards play complementary temporal roles: diversity broadens early exploration, whereas efficiency converts that exploration into selective harness use as training progresses. This interpretation also complements Theorem 1: the bounded diversity bonus adds at most λ_u to the final cost-aware objective gap and vanishes from the objective after annealing. This does not rule out optimization-path effects from early exploration.

The early role of NOTE. Although the evolver receives complete trajectories, useful discoveries and corrections can be difficult to identify in long interaction histories. NOTE lets the acting policy mark this evidence when it is most salient, giving consolidation a sparse signal about what should be added or revised. Its relatively frequent use early in training therefore helps the initially empty Experience bank accumulate useful knowledge quickly. As the bank matures and efficiency pressure increases, NOTE calls decline, while the Experience distilled from earlier notes remains available for later retrieval. Thus, even though NOTE is invoked less often later in training, its more frequent early use still provides important support for bootstrapping the Experience bank used in subsequent coordination.

B.4 FROM INTERFACE ACQUISITION TO SELECTIVE COORDINATION

Figure 9 compares harness usage before and after coordination training. Here, Qwen3-8B BASE denotes the pretrained policy placed directly in the adaptive BPE action interface, rather than the

always-on EVOHARNESS-BASE scaffold. Despite having access to the interface, the base policy invokes harness actions only sparsely, suggesting that prompting alone is insufficient for reliably operating the newly introduced meta-action space. SFT addresses this cold-start problem by imitating successful teacher trajectories in which the BPE operations are explicitly demonstrated, substantially increasing harness use together with task success. After this interface has been acquired, reinforcement learning reverses the trend: EVOHARNESS-RL (Evo-RL) reduces the number of harness calls while success continues to improve across all three environments. The resulting pattern suggests that SFT primarily teaches the policy *how to use* the harness, whereas RL learns *when its use is worth the interaction cost*, pruning redundant calls while retaining task-relevant external support.

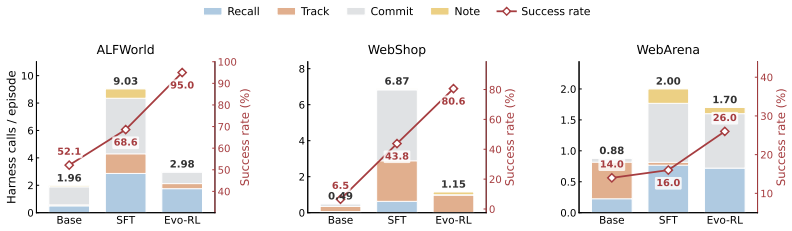

Figure 9: Average harness calls per test episode and test-set success rates across the three benchmarks.

C CROSS-DOMAIN TRANSFERABILITY

We further test whether the coordination learned by EVOHARNESS-RL transfers across environments by initializing ALFWorld training from a WebShop-trained checkpoint and adapting for 100 target-domain steps. As shown in Figure 10, EVOHARNESS-RL transfer adapts rapidly and progressively approaches the in-domain EVOHARNESS-RL policy on both training and validation performance. In contrast, transferring a standard GRPO checkpoint exhibits strong negative transfer: training performance partially recovers, but validation success remains substantially lower throughout adaptation. Removing target-side SFT also degrades transfer, indicating that interface initialization still facilitates adaptation. Together, these results suggest that EVOHARNESS-RL learns coordination patterns that are partially reusable across domains, providing a better initialization than transferring environment-specific policy behavior alone.

Interestingly, cross-domain transfer does not imply that the same harness-use pattern should be preserved across environments. Our earlier analysis shows that WebShop and ALFWorld converge to different mixtures of BPE operations. What appears to transfer is instead the underlying coordination policy: how the agent decides whether external support is useful under the current execution state. The observed action frequency is environment dependent, since different domains induce different state and information demands, while the conditional decision rule for when to externalize state can remain reusable. This helps explain why EVOHARNESS-RL exhibits substantially less negative transfer than standard GRPO: it transfers a reusable coordination prior rather than a fixed source-domain action pattern.

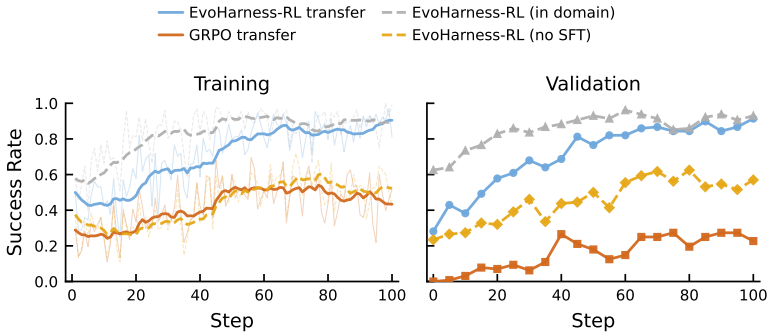

Figure 10: Cross-domain adaptation from WebShop to ALFWorld over 100 target-domain training steps. Left: training success. Right: validation success.

D CROSS-BACKBONE ADAPTABILITY

We evaluate whether our framework remains effective across backbone scales. Table 5 reports selected results with Qwen3-4B on ALFWorld, WebShop, and WebArena, alongside Qwen3-8B EVOHARNESS-RL as a reference. This comparison tests the adaptability of our harness-learning framework beyond the 8B backbone used in the main experiments. Unreported results are marked with an em dash.

Table 5: Cross-backbone adaptability across ALFWorld, WebShop, and WebArena.

Backbone	Approach	ALFWorld							WebShop		WebArena				
		Pick	Look	Clean	Heat	Cool	Pick2	Avg.	SR	Score	SP	CMS	Maps	G&R	Overall
4B	ReAct ∗	62.9	61.5	11.1	62.5	8.0	8.3	33.6	3.2	39.5	13.3	6.7	10.0	0.0	8.0
	SkillRL 🔥	94.3	69.2	85.2	68.8	92.0	87.5	85.7	74.4	84.0	26.7	20.0	0.0	10.0	16.0
	EVOHARNESS-BASE ∗	85.7	30.8	51.9	50.0	36.0	33.3	52.1	12.6	28.7	20.0	6.7	20.0	30.0	18.0
	EVOHARNESS-SFT 🔥	57.1	7.7	37.0	12.5	12.0	16.7	28.6	39.0	46.3	20.0	6.7	0.0	10.0	10.0
	EVOHARNESS-RL 🔥	100.0	61.5	92.6	81.3	80.0	91.7	87.9	79.0	89.0	20.0	20.0	10.0	20.0	18.0
8B	EVOHARNESS-RL 🔥	**100.0**	**92.3**	**96.3**	**100.0**	**84.0**	**95.8**	**95.0**	**80.6**	**90.8**	**40.0**	**20.0**	**10.0**	**30.0**	**26.0**

E METHOD ROBUSTNESS ACROSS RANDOM SEEDS

Table 1 reports a single evaluation run, whereas here we evaluate EVOHARNESS-RL across three random seeds and report the mean and population standard deviation in Table 6. Randomness has little effect on ALFWorld, where every run solves 133 of 140 tasks, or on WebShop, which achieves $80.5 \pm 0.7\%$ success and an average score of 89.8 ± 0.9. WebArena exhibits greater variation partly because the standardized AgentGym-RL split is comparatively small: it contains 372 training tasks and only 50 evaluation tasks, divided into Shopping (15), CMS (15), Maps (10), GitLab (5), and Reddit (5). A single changed outcome can therefore move a domain score by 6.7–20 percentage points, making the aggregate result over all 50 tasks more informative than any individual category. The three overall scores are 20.0%, 22.0%, and 26.0%, yielding $22.7 \pm 2.5\%$. The gains on WebArena relative to ReAct also vary across task categories: they are strongest on Shopping and GitLab/Reddit, while CMS and Maps benefit less. The latter tasks depend more on precise site-specific action execution and current-page grounding, leaving less room for persistent state and reusable Experience to help. Because of time and compute constraints, we run three seeds only for EVOHARNESS-RL; the comparison baselines reproduce the point estimates from Table 1. Nevertheless, even the mean minus one standard deviation for our method exceeds the strongest listed baseline in 11 of the 14 reported columns.

Table 6: Robustness across three random seeds. Baselines reproduce the point estimates from Table 1; EVOHARNESS-RL reports mean ± population standard deviation over three runs. All values are percentages except WebShop Score. Bold indicates the best result in each column.

Approach	ALFWorld							WebShop		WebArena				
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	SR	Score	SP	CMS	Maps	G&R	Avg.
ReasoningBank ∗	83.8	48.7	49.4	39.6	41.3	54.2	56.2	11.4	35.4	20.0	13.3	**10.0**	10.0	14.0
SkillOS-base ∗	79.0	41.0	45.7	37.5	38.7	55.6	53.1	13.6	38.6	20.0	**20.0**	0.0	20.0	16.0
SkillOS 🔥	85.7	56.4	54.3	43.8	46.7	62.5	61.2	16.5	40.6	26.7	**20.0**	**10.0**	10.0	18.0
SkillRL 🔥	94.3	84.6	92.6	87.5	72.0	75.0	85.0	73.2	85.5	20.0	**20.0**	**10.0**	20.0	18.0
EVOHARNESS-RL 🔥	**100.0** ± 0.0	**94.9** ± 3.6	**96.3** ± 0.0	**100.0** ± 0.0	**82.7** ± 1.9	**95.8** ± 0.0	**95.0** ± 0.0	**80.5** ± 0.7	**89.8** ± 0.9	**31.1** ± 6.3	**20.0** ± 0.0	6.7 ± 4.7	**30.0** ± 0.0	**22.7** ± 2.5

F THEORETICAL ANALYSIS

This appendix analyzes what the two training stages in Section 2.3 can guarantee, in the order in which they act. Supervised initialization yields a policy that can operate the BPE interface, so Appendix F.1 first fixes such a policy and proves Proposition F.1: when a BPE return carries decision-relevant information that the policy uses with bounded error, fewer independent rollouts suffice to

discover a successful trajectory with high probability. Reinforcement learning then updates the policy under the scheduled reward of Equation (5), whose diversity bonus is annealed. Appendix F.2 proves Theorem 1: the bonus shifts the final cost-aware objective by at most its annealed weight, so after annealing, the probability of selecting an optimal coordination rule is limited only by the optimization residual. Both results are conditional, and Appendix F.3 states what they do not establish.

F.1 Conditional Rollout Efficiency for Success Discovery

This subsection proves Proposition F.1, which compares how many independent rollouts a baseline policy and a BPE policy need to discover a successful trajectory from the same context. The proof has three steps: BPE information raises the probability that the next search finds its target (Step 1), a higher hit probability raises full-task success (Step 2), and higher success reduces the required number of rollouts (Step 3). A case from the paper first identifies the decision; we then fix the comparison, define the quantities of each step, and prove the result.

The analyzed decision. The kettle-cleaning episode in Appendix H contains the decision analyzed here. After committing to find the kettle, the agent calls recall, which suggests a countertop, and must then choose where to search. The suggestion is wrong, yet the agent checks the stoveburners and completes the task. A faithful model must therefore allow a return to mislead and a search miss to be followed by recovery; the episode illustrates the decision but measures none of the parameters below.

Comparison and assumptions. Let x be the policy-visible context at such a search decision, before the analyzed BPE return, following the prompt notation x_t. We compare two fixed checkpoints, a baseline policy π_0 and a BPE policy π_1; index 0 refers to the baseline and index 1 to the BPE policy throughout. Both use the same experience bank, held fixed as within a rollout batch (Section 2.1), and the same remaining interaction budget H_x, which BPE calls consume together with environment actions. A rollout runs from x to termination, and the comparison rests on two assumptions.

(A1) *Independent attempts.* Each rollout independently draws a task state from the same conditional distribution given x and samples its actions independently; any hidden state held fixed across attempts is included in x.

(A2) *Common coupling.* Both policies are analyzed on the same joint distribution of task state and BPE return M given x, and reading M changes neither the target state nor the legal search choices. The baseline chooses its search action independently of M given x, whereas the BPE policy may condition on M.

Under (A2), the same search choice has the same hit probability for both policies, so any difference in discovery comes from how each policy uses the return and from what happens after the search.

Step 1 quantities: search hits. Let m denote a value of M, with $w_x(m) = \Pr(M = m \mid x)$; an empty return covers failed retrieval or no call, and sums become integrals for continuous returns. Let $\mathcal{A}_x$ be the finite nonempty set of legal search choices. For $a \in \mathcal{A}_x$, $h_x(m, a)$ is the true probability that a finds the target given x and m, averaged over the remaining environment uncertainty; it is an environment probability, not a confidence reported by the model. With the BPE policy's choice probabilities $\pi_1(a \mid x, m)$, define

$$v(x, m) = \max_{a \in \mathcal{A}_x} h_x(m, a), \qquad q(x, m) = \sum_{a \in \mathcal{A}_x} \pi_1(a \mid x, m) h_x(m, a), \tag{8}$$

$$e(x, m) = v(x, m) - q(x, m) \in [0, 1]. \tag{9}$$

Here v is the best hit probability achievable with the same information, q is the policy's actual hit probability, and e is its decision error. The two policies' hit probabilities are

$$s_0(x) = \sum_a \pi_0(a \mid x) \sum_m w_x(m) h_x(m, a), \qquad s_1(x) = \sum_m w_x(m) q(x, m). \tag{10}$$

Their difference depends on how much the return can improve the search and on how closely the policy realizes that improvement.

Information value and decision error. Without the return, the best hit probability is $v_0(x) = \max_a \sum_m w_x(m)h_x(m,a)$, and the information value is

$$\gamma(x) = \sum_m w_x(m)v(x,m) - v_0(x) \geq 0. \tag{11}$$

It is nonnegative because a decision-maker holding the return can ignore it, and it is positive only when some return changes the best search choice. How closely the policy realizes this value is measured at a permitted failure fraction $0 \leq \delta < 1$ by the conditional quantile of its decision error,

$$F_x(z) = \sum_m w_x(m)\mathbf{1}\{e(x,m) \leq z\}, \qquad \varepsilon_\delta(x) = \inf\{z \in [0,1] : F_x(z) \geq 1-\delta\}, \tag{12}$$

and set $b_\delta(x) = (1-\delta)\varepsilon_\delta(x) + \delta$, which Step 1 of the proof shows to bound the mean decision error. The failure fraction δ concerns returns, whereas the confidence ρ in Step 3 concerns rollouts. For finitely many returns, $\varepsilon_\delta(x)$ is read off by sorting the errors and accumulating their probabilities until the mass reaches $1-\delta$. The quantile guarantees $\Pr(e(x,M) \leq \varepsilon_\delta(x) \mid x) \geq 1-\delta$ by construction, so the substantive capability condition is that $\varepsilon_\delta(x)$ is small. This approximate-use model follows the Bayesian view of in-context learning in Xie et al. (2022); Zhang et al. (2025b), and experience-reuse results with Qwen3-8B in ReMe and SkillOS motivate it empirically (Cao et al., 2026; Ouyang et al., 2026).

Step 2 quantities: from search hits to task success. A search hit matters only through its effect on completing the task. For $i \in \{0,1\}$, let $r_{i,h}(x)$ and $r_{i,m}(x)$ be policy i's full-task completion probabilities given that its next search hits or misses, under the same remaining budget; $r_{i,m}$ accounts for recovery after a miss, as in the kettle episode. Policy i then succeeds with probability

$$p_i(x) = s_i(x)r_{i,h}(x) + \big(1 - s_i(x)\big)r_{i,m}(x). \tag{13}$$

Two quantities convert a hit into success:

$$g(x) = r_{0,h}(x) - r_{0,m}(x), \tag{14}$$

$$\kappa(x) = s_1(x)\big[r_{0,h}(x) - r_{1,h}(x)\big] + \big(1 - s_1(x)\big)\big[r_{0,m}(x) - r_{1,m}(x)\big], \tag{15}$$

$$c(x) = \max\{\kappa(x), 0\}. \tag{16}$$

The gain g is the baseline's completion advantage of a hit over a miss. The loss c measures, in units of task-success probability, the budget spent on calls and any change in subsequent execution; any specified upper bound on κ may replace it. The quantities of Steps 1 and 2 combine into the net benefit and the condition of Proposition F.1:

$$\begin{gathered} \alpha(x) = g(x)\big[\gamma(x) - b_\delta(x)\big] - c(x) > 0, \\ g(x) > 0, \qquad 0 < s_i(x) < 1, \qquad 0 < p_0(x) < p_0(x) + \alpha(x) < 1. \end{gathered} \tag{17}$$

The condition requires that useful information, after decision error, raises completion by more than the extra loss.

Proposition F.1 (Conditional rollout efficiency for success discovery)**.** *Let $N_{\text{base}}(x)$ and $N_{\text{BPE}}(x)$ be the minimum numbers of independent rollouts needed to find at least one successful trajectory conditioned on context x with high probability, that is, with probability at least $1-\rho$ for a common $0 < \rho < 1$. Under (A1), (A2), and Condition (17), the BPE policy needs at most a $1/(1+\alpha(x)/p_0(x))$ fraction of the baseline's rollouts, rounded up, to find a successful trajectory with high probability:*

$$N_{\text{BPE}}(x) \leq \left\lceil \frac{1}{1+\alpha(x)/p_0(x)} \cdot N_{\text{base}}(x) \right\rceil, \tag{18}$$

where $p_0(x) \in (0,1)$ is the baseline task completion probability and $\alpha(x) > 0$ is the conservative net benefit from BPE information defined above.

Proof. Fix x and suppress it in the notation.

Step 1 (search hits). Let $\beta = \Pr(e > \varepsilon_\delta) \leq \delta$. Since $0 \leq e \leq 1$ and $\varepsilon_\delta \leq 1$, $\mathbb{E}[e] \leq (1-\beta)\varepsilon_\delta + \beta \leq b_\delta$. By (A2), the baseline's choice is independent of M, so $s_0 = \sum_a \pi_0(a) \sum_m w(m)h(m,a) \leq v_0$. With $q = v - e$,

$$s_1 - s_0 \geq \mathbb{E}_M[v] - v_0 - \mathbb{E}_M[e] \geq \gamma - b_\delta. \tag{19}$$

Step 2 (task success). Adding and subtracting $s_1 r_{0,h} + (1 - s_1) r_{0,m}$ in Equation (13) and using $g > 0$,

$$p_1 - p_0 = g(s_1 - s_0) - \kappa \geq g(\gamma - b_\delta) - c = \alpha > 0. \tag{20}$$

Step 3 (rollout count). By (A1), n rollouts with success probability $0 < p < 1$ all fail with probability $(1-p)^n$, so the minimum number that discovers a success with probability at least $1 - \rho$ is

$$N(p) = \left\lceil \frac{\log(1/\rho)}{f(p)} \right\rceil, \qquad f(p) = -\log(1-p), \tag{21}$$

and $N_{\text{base}} = N(p_0)$. Let $k = 1 + \alpha/p_0$. Since f is increasing and convex with $f(0) = 0$, and $kp_0 = p_0 + \alpha < 1$, $f(p_1) \geq f(p_0 + \alpha) \geq k f(p_0)$. For $n = \lceil N_{\text{base}}/k \rceil$,

$$n f(p_1) \geq N_{\text{base}} f(p_0) \geq \log(1/\rho), \tag{22}$$

so n BPE rollouts suffice, proving Equation (18). If $p_1 = 1$, one rollout suffices and the bound holds trivially. □

F.2 Conditional Convergence with Diversity Annealing

Proposition F.1 holds the policy fixed, whereas training updates it under a scheduled reward that adds a temporary diversity bonus to the final objective. This subsection proves Theorem 1, which bounds how far the epoch-u policy is from the optimum of the final cost-aware objective and how likely it is to select an optimal coordination rule. We define the rule class and the two objectives, restate the theorem, prove it in two steps, and identify when its residual vanishes.

Rules and objectives. Let $\mathcal{B}$ be a finite class of complete, context-dependent coordination rules; each rule specifies harness calls together with the subsequent execution decisions, rather than a single harness action. We hold the task distribution, experience bank, interaction horizon, and efficiency weight fixed, so the final objective does not depend on the epoch. For a rule b, the final cost-aware objective and the diversity score are

$$J(b) = \mathbb{E}_b \left[R_{\text{GRPO}}(\tau) + \lambda_{\text{eff}} R_{\text{eff}}(\tau) \right], \qquad d(b) = \mathbb{E}_b[R_{\text{div}}(\tau)] \in [0,1], \tag{23}$$

where $d(b) \leq 1$ because R_{div} is the fraction of the four BPE action types used. The efficiency term is active throughout training; only diversity is annealed, with weight $\lambda_u = \lambda_{\text{div}}(u) \geq 0$ at epoch u. The policy at epoch u induces a distribution μ_u over $\mathcal{B}$, and for any distribution μ we write $J(\mu) = \sum_b \mu(b) J(b)$ and $d(\mu) = \sum_b \mu(b) d(b)$. Training at epoch u thus targets the scheduled objective $J(\mu) + \lambda_u d(\mu)$, whose optimization residual is

$$\varepsilon_{\text{opt}}(u) = \max_{b \in \mathcal{B}} \big[J(b) + \lambda_u d(b) \big] - \big[J(\mu_u) + \lambda_u d(\mu_u) \big] \geq 0. \tag{24}$$

The residual is defined at every finite epoch; it presupposes neither that the optimizer converges nor that the neural policy can represent an optimal rule. The final objective determines the optimal rules and their margin,

$$J^* = \max_{b \in \mathcal{B}} J(b), \qquad \mathcal{B}^* = \{ b : J(b) = J^* \}, \qquad \Delta_J = \min_{b \notin \mathcal{B}^*} \big[J^* - J(b) \big] > 0, \tag{25}$$

where finiteness makes Δ_J well defined unless every rule is optimal, in which case the theorem is trivial. Since $b \sim \mu_u$ in the main text, $\Pr(b \in \mathcal{B}^*) = \mu_u(\mathcal{B}^*)$, and the theorem reads as follows.

Theorem 1 (restated). *For every epoch u,*

$$J^* - J(\mu_u) \leq \varepsilon_{\text{opt}}(u) + \lambda_u \quad \textit{and} \quad \mu_u(\mathcal{B}^*) \geq 1 - \frac{1}{\Delta_J} \big(\varepsilon_{\text{opt}}(u) + \lambda_u \big). \tag{6) and (7}$$

In particular, $\lambda_u = 0$ for all $u \geq U$, so if $\varepsilon_{\text{opt}}(u) \to 0$, then $J(\mu_u) \to J^*$ and $\mu_u(\mathcal{B}^*) \to 1$.

Proof. *Step 1 (objective gap).* Choose $b^* \in \mathcal{B}^*$. The scheduled objective of μ_u is within $\varepsilon_{\text{opt}}(u)$ of its maximum, and this maximum is at least the scheduled objective of b^*. Hence

$$J^* + \lambda_u d(b^*) \leq J(\mu_u) + \lambda_u d(\mu_u) + \varepsilon_{\text{opt}}(u), \tag{26}$$

$$J^* - J(\mu_u) \leq \varepsilon_{\text{opt}}(u) + \lambda_u \big[d(\mu_u) - d(b^*) \big] \leq \varepsilon_{\text{opt}}(u) + \lambda_u, \tag{27}$$

where the last inequality uses $d(\mu_u) \le 1$ and $d(b^*) \ge 0$. This proves Equation (6).

Step 2 (optimal rules). Every non-optimal rule loses at least Δ_J in the final objective, so

$$\Delta_J\big[1 - \mu_u(\mathcal{B}^*)\big] \le \sum_b \mu_u(b)\big[J^* - J(b)\big] = J^* - J(\mu_u). \tag{28}$$

Dividing by Δ_J and applying Step 1 proves Equation (7). When $\lambda_u = 0$ and $\varepsilon_{\mathrm{opt}}(u) \to 0$, both bounds give the limits stated in the theorem. □

When the residual vanishes. The cosine schedule in Section 2.3 sets $\lambda_u = 0$ for all $u \ge U$, so after annealing the residual reduces to $\varepsilon_{\mathrm{opt}}(u) = J^* - J(\mu_u)$. Its vanishing is therefore an optimization-consistency condition on the fixed rule class, $J(\mu_u) \to J^*$, which concerns the objective value; convergence to a stationary point does not suffice. Exact-gradient results in finite discounted MDPs motivate this condition: Agarwal et al. (2021, Theorem 10) for tabular softmax policy gradient with a full-support state distribution and a prescribed step size, and Xiao (2022, Theorem 8) for policy mirror descent under direct parameterization with interior initialization and a constant step size. Our training additionally regularizes toward the SFT reference and adds an entropy bonus (Appendix K); any persistent gap they create, together with optimization and representation error, remains in $\varepsilon_{\mathrm{opt}}(u)$, and the finite-epoch bounds hold regardless. After annealing, retained harness use is therefore determined only by task reward, interaction efficiency, and the optimization residual, which is the sense of convergence in Theorem 1.

F.3 Scope of the Two Results

Proposition F.1 and Theorem 1 each certify one link between the training design and the empirical results, under the conditions stated above. This subsection lists what they do not establish, so that the corresponding claims in the main text and in Appendix B are read at their intended strength.

Fixed-policy discovery is not training sample complexity. Proposition F.1 compares two fixed checkpoints with a fixed bank at one context, so it does not bound the number of GRPO steps. The speedups in Appendix B are measured; the proposition supplies a mechanism consistent with them, not a prediction of their size. Its integer form has two edge cases: rounding may make both budgets equal, and when $p_0 = 0$ the baseline has no finite discovery budget.

The gain requires decision-relevant information. Even at a fixed checkpoint, the condition $\alpha > 0$ is an assumption about the return and the policy, not a consequence of calling BPE. If M deterministically rewrites information already in x, then $\gamma = 0$ and the condition cannot certify a gain; a benefit from reorganizing the same information would instead require comparing the two policies' processing errors. The loss c is defined through completion probabilities rather than derived from call counts. The analysis assumes neither context truncation, perfect retrieval, nor failure after a search miss.

Theorem 1 is neither a rate nor a comparison with GRPO. On the training side, the theorem bounds the objective gap by the current residual but does not show that early diversity makes this residual shrink faster than standard GRPO, nor that diversity leaves the optimization path unchanged; the benefit of early exploration is supported by the reward ablations in Appendix B.3. It relies on a finite rule class with a positive margin Δ_J, and its probability refers to the current policy selecting any optimal rule, not to a training run finding globally optimal parameters.

Fixed banks and cited results. Both analyses hold the experience bank fixed, so neither credits a note for improving future banks through consolidation. The cited works motivate the modeling conditions without verifying them here: the high-probability bounds of Zhang et al. (2025b) concern pretraining data rather than BPE returns, ReMe and SkillOS do not estimate δ, ε_δ, or α, and the policy-gradient results cover exact gradients in tabular or directly parameterized settings rather than neural GRPO.

G More Ablation Analysis

Learned coordination beyond harness frequency. We further test whether the learned coordination policy captures more than the overall amount of harness use. We evaluate the same final checkpoint with the same frozen Experience bank and decoding protocol, changing only the schedule of harness interactions. As shown in Table 7, the learned schedule achieves the strongest overall performance. Shuffling invocation timing causes a moderate drop, while shuffling harness-action types leads to a substantially larger degradation. A frequency-matched control also remains well below the learned policy despite preserving a comparable amount of harness use. These results indicate that the gain does not come simply from invoking the harness more often: the policy learns which external operation is useful in the current execution state, with action selection appearing more important than precise invocation timing on ALFWorld. The effect is strongest on task families requiring state disambiguation or multi-stage coordination, while simple pick-and-place tasks are largely insensitive to schedule perturbations.

Table 7: ALFWorld harness-schedule ablation under a shared checkpoint, Experience bank, and decoding protocol. Δ is the drop from Learned.

Variant	ALFWorld							Δ
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	
Learned	100.0	92.3	96.3	100.0	84.0	95.8	**95.0**	–
Timing Shuffle	100.0	100.0	96.3	87.5	84.0	91.7	93.6	▼1.4
Type Shuffle	100.0	61.5	92.6	100.0	76.0	87.5	88.6	▼6.4
Freq. Matched	100.0	76.9	96.3	93.8	76.0	91.7	90.7	▼4.3

Table 8: ALFWorld BPE component ablation for EvoHarness-Base under a shared evaluation protocol. Δ is the drop from Full BPE.

Variant	ALFWorld							Δ
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.	
Full BPE	71.4	53.8	63.0	50.0	48.0	41.7	56.4	–
w/o Belief	74.3	53.8	40.7	75.0	20.0	37.5	50.0	▼6.4
w/o Progress	68.6	53.8	44.4	68.8	32.0	37.5	50.7	▼5.7
w/o Experience	65.7	53.8	40.7	62.5	28.0	41.7	48.6	▼7.8

Inference-time BPE components. Table 8 isolates the contribution of each component in the prompt-time harness. Removing Belief most strongly affects tasks that require localization and state verification, while removing Progress harms dependent-subgoal tasks such as Pick2. Removing Experience yields the lowest overall average, confirming that all three state types provide complementary support at inference time.

RL internalizes harness-use patterns beyond explicit instructions. Table 9 shows that sensitivity to explicit harness instructions decreases substantially after RL optimization. Removing guidance for when and how to invoke harness actions reduces EvoHarness-SFT from 68.57% to 59.29%, a drop of 9.29 percentage points. In contrast, EvoHarness-RL remains largely robust, decreasing only from 95.00% to 92.86% (-2.14 points). This suggests that SFT still relies on explicit prompt guidance to coordinate with the external harness, whereas RL increasingly internalizes when external state should be accessed. Together with harness annealing, where explicit harness calls decrease while success improves, this provides complementary evidence that EvoHarness-RL learns the coordination strategy itself rather than simply following hand-written instructions.

Table 9: Robustness to removing explicit harness-use instructions on the 140-task ALFWorld seen split.

Policy	Stripped SR	Full-Prompt SR	Drop
EvoHarness-SFT 🔥	59.29	68.57	↓ 9.29
EvoHarness-RL 🔥	**92.86**	**95.00**	↓ **2.14**

H Qualitative Case Study

We present two complementary ALFWorld cases that expose the self-evolving harness at different timescales: policy–harness coordination within one episode and Experience consolidation across episodes.

Harness-guided reasoning within an episode. In the first case, the agent must clean a kettle and place it on the dining table. The trajectory shows a natural commit → recall → act → note loop. The agent commits to a search subgoal, recalls task and object-location priors, verifies them through interaction, and writes corrective evidence when the recalled hint is stale. Rather than repeatedly searching the suggested countertops, it finds the kettle on a stoveburner and records the contradiction for future consolidation.

Case Study: Skill Loop with Self-Correction

Task: clean some kettle and put it in diningtable.

Type: pick_clean_then_place_in_recep **SUCCESS**: score 1.0, 30 turns

Skills	**[cle_001]** Phase-ordered plan: locate → clean → place; **[cle_006]** object-location priors.
Avoid	**[err_002]** Blindly trusting stale hints; **[err_003]** repeatedly searching the same locations.

Agent Trajectory with Harness-Guided Reasoning

Step 1	**Obs:** Initial task received. **Think:** Register the first subgoal before acting; decompose the task into locate, clean, and place. **Harness:** **commit** [find kettle]
Step 2	**Obs:** The agent needs a search strategy for the kettle. **Think:** Recall both the general clean-task procedure and likely kettle locations. **Harness:** **recall** [how to do clean task] → **recall** [where to find kettle] **Return:** kettle: countertop; avoid revisiting the same failed locations.
Step 3	**Obs:** countertop 1 empty; countertop 2 empty; stoveburner 1 has only a pan; stoveburner 2 has only a pot; stoveburner 3 contains the kettle. **Think:** The recalled countertop hint fails twice. Following **[err_003]**, the agent avoids looping and falls back on the prior that kettles are often on stoveburners. **Action:** go to countertop 1 → go to countertop 2 → go to stoveburner 3.
Step 4	**Obs:** The kettle is found at stoveburner 3, contradicting the recalled countertop prior. **Think:** Use the current trajectory as corrective evidence instead of blindly trusting stale memory. **Action:** take kettle 1 from stoveburner 3. **Harness:** **note** [kettle found at stoveburner 3; recalled hint said countertop].
Step 5	**Obs:** The kettle is held by the agent. **Think:** Continue the phase-ordered clean task: clean the kettle, then place it on the dining table. **Harness:** **commit** [clean kettle with sinkbasin]. **Action:** go to sinkbasin 1 → clean kettle 1 with sinkbasin 1 → go to diningtable 1 → move kettle 1 to diningtable 1.

Result: **SUCCESS**. The trajectory exhibits the full harness loop: **commit** → **recall** → act → **note**. The recalled hint was wrong; the agent detected the contradiction, adapted, and wrote corrective evidence back to the skill bank.

Experience evolution across episodes. The second case follows that corrective evidence beyond a single rollout. It traces how an initially incorrect cooling procedure enters the Experience bank, causes a failed trajectory, is corrected by the evolver, and is finally abstracted into a reusable general workflow.

Case Study: Experience Evolution through Self-Correction

Task family: pick_cool_then_place_in_recep **Entries:** `pic_005` (task-specific), `gen_015` (general)

Bank denotes persistent Experience available to later rollouts; gray text summarizes the evidence and reason for each edit.

End-to-End Skill-Evolution Trajectory

Step 1 **Cycle 1 — Add an initial skill.**
Bank before: *No cooling procedure.*
Evidence contains only object-location notes; the bank lacks reusable cooling guidance.
Operation: ADD `pic_005`.
Bank after: *Take the object to a fridge, put it in, close the fridge, then retrieve it.*
The inferred rule is invalid but is retrieved 127 times by step 10.

Step 2 **Cycle 11 — Failure reveals the error.**
Bank before: *Put the object inside the fridge.*
Rollout: find plate → take plate → go to fridge → **put plate in fridge** → repeat fridge navigation → **FAILURE**.
Insertion never triggers ALFWorld's cooling transition; the failed trajectory is buffered as correction evidence.

Step 3 **Cycle 11 — Correct the persistent entry.**
Bank before: *Put it in, close the fridge, then retrieve it.*
Operation: UPDATE `pic_005`.
Bank after: *Hold the object and issue `cool object with fridge`; never put it inside the fridge.*
The update preserves the entry's existing `usage_count` of 145.

Step 4 **Cycle 25 onward — Generalize and reuse.**
Bank before: *Correct, but task-specific `pic_005`.*
The corrected mechanism applies across objects and destinations.
Operation: ADD general entry `gen_015`.
Bank after: *Take → cool while holding → move to destination.*
Reuse: recall bread location → take bread → cool with fridge → move to diningtable → **SUCCESS**.
The generalized entry reaches 882 retrievals by step 150.

Evolution loop: rollout evidence → ADD → failure feedback → UPDATE → global abstraction → successful reuse. The example shows that Experience evolution changes both the content and scope of a skill rather than merely appending new notes.

What the cases show. Together, the cases reveal a closed feedback loop between policy behavior and external Experience. Within an episode, the policy treats recalled knowledge as a revisable prior: it grounds the suggestion against current observations and emits note when the two disagree. Across episodes, the evolver turns these notes and rollout outcomes into bank edits, correcting the faulty rule while preserving its retrieval history and then lifting the corrected task-specific procedure into a general skill. The updated skill subsequently guides a shorter, successful trajectory. Thus, self-evolution is not merely the accumulation of additional memory entries; it is an evidence-driven loop in which the policy tests Experience, reports failures, and benefits from the reconciled and abstracted Experience in later rollouts.

I ENVIRONMENT-SPECIFIC BPE GROUNDING

BPE specifies a shared functional interface, while each environment adapter determines how the corresponding state is represented and maintained. In EVOHARNESS-BASE, the adapter updates BPE from agent-visible observations and action outcomes and automatically renders the resulting compact workspace at each step. The policy selects only environment actions, so access to this always-on workspace consumes no additional interaction step. In EVOHARNESS-SFT and EVOHARNESS-RL, the same environment-grounded stores support adaptive coordination: BPE content is exposed or updated only when the policy emits the corresponding meta-action, and each such call consumes one interaction step. Belief and Progress are maintained within an episode, whereas Experience per-

sists and evolves across episodes. No variant exposes privileged environment state; all policy-facing content is derived from normal agent interaction.

I.1 Environment-Specific State Grounding

Belief (B_t) and track. Belief provides persistent within-episode state that would otherwise disappear from the immediate observation context. In ALFWorld, we maintain a lightweight world-state store from action-observation pairs, including object states, locations, receptacle relations, and inventory information. In WebShop, the adapter maintains shopping state derived from the user request and product information observed during search and comparison. In WebArena, Belief is implemented as a compact application-state representation constructed from the agent-visible accessibility tree and interaction history. It preserves the current page, navigation history, active controls, and task-relevant records collected from earlier pages without accessing the raw DOM or environment ground truth. The underlying Belief stores are updated after every environment interaction. EvoHarness-Base renders their compact views automatically, whereas in the adaptive setting track performs a selective read without changing Belief: track `[object]` returns matching object states and relations in ALFWorld, while WebShop routes track `[price]`, track `[candidates]`, and track `[options]` to budget, comparison, and variant-selection views. In WebArena, its arguments select navigation history, previously observed values, objective-relevant text, or a literal match from earlier pages. The returned slice is inserted into the visible harness context for the next policy decision.

Progress (P_t) and commit. Progress externalizes the execution state of the current task. In ALFWorld, it is represented as a bounded subgoal record over household manipulation steps. In WebShop, it tracks which product requirements have been resolved and which remain to be verified before purchase. In WebArena, it maintains completed and pending requirements across multi-page workflows. EvoHarness-Base automatically maintains and displays this execution summary from observed outcomes. In the adaptive setting, commit `[subgoal]` writes the current subgoal to the bounded Progress record and returns the updated plan, so a later decision can recover what has been completed and what remains without reconstructing the full trajectory.

Experience (E_t), recall, and note. Experience stores reusable cross-episode knowledge. We organize it into **general experience**, which captures broadly reusable interaction principles, and **task-specific experience**, which captures procedures and failure patterns associated with particular task families. Individual adapters may additionally maintain compact mistake-correction entries or domain-specific search priors. EvoHarness-Base automatically retrieves and displays relevant entries at each step. In the adaptive setting, recall `[query]` instead performs this read on demand: the query selects relevant general principles, task-specific procedures, mistakes, or search hints and inserts them into the visible context. Conversely, note `[insight]` is a write to a worker-local evidence buffer. It records a reusable discovery or a correction to recalled Experience, but does not modify the persistent bank during the rollout; buffered notes are validated against outcomes and reconciled only during batched consolidation. Although the consolidator also receives completed trajectories and the current Experience snapshot, a reusable mismatch can be difficult to identify in a long trace: the recalled recommendation and the later observation that confirms or contradicts it may be separated by many unrelated interactions. At the moment of the mismatch, the acting policy has both pieces in its visible context, so note provides a sparse, localized annotation of what may need to be added or corrected and why. It is only a proposal rather than an authoritative memory write; the consolidator checks it against trajectory outcomes and other batch evidence before choosing Add, Update, Archive, or Skip.

I.2 Experience Evolution and Consolidation

We separate online task execution from cross-episode Experience evolution. During rollout collection, the policy acts as the *solver* using a fixed snapshot of the current Experience store. Completed trajectories, outcomes, and note outputs are accumulated as evidence. The *evolver* operates outside the rollout loop: it abstracts reusable general and task-specific Experience from this evidence and reconciles the resulting candidates with the existing store through four operations: Add, Update, Archive, and Skip.

Algorithm 1 Batched Experience Evolution

Require: Experience store $\mathcal{E}^{(k)}$, policy π_θ, rollout batch $\mathcal{D}^{(k)}$, workers $w_i{}_{i=1}^{W}$
Ensure: Updated Experience store $\mathcal{E}^{(k+1)}$

$\bar{\mathcal{E}} \leftarrow \mathrm{Freeze}(\mathcal{E}^{(k)})$
$\mathcal{B} \leftarrow \emptyset$
for all workers w_i **in parallel do**
  $\mathcal{B}_i \leftarrow \emptyset$
  for all $x \in \mathcal{D}^{(k)} * i$ **do**
    $(\tau, y, N) \leftarrow \mathrm{Rollout}(\pi * \theta, x, \bar{\mathcal{E}})$
    $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup (x, \tau, y, N)$
  end for
end for
$\mathcal{B} \leftarrow \bigcup_{i=1}^{W} \mathcal{B} * i$
$\mathcal{C} * \mathrm{task} \leftarrow \mathrm{AbstractTaskExperience}(\mathcal{B})$
$\mathcal{C} * \mathrm{gen} \leftarrow \mathrm{AbstractGeneralExperience}(\mathcal{B})$
$\mathcal{C} \leftarrow \mathcal{C} * \mathrm{task} \cup \mathcal{C}_{\mathrm{gen}}$
$\Delta \leftarrow \mathrm{Reconcile}(\mathcal{C}, \bar{\mathcal{E}})$ ▷ ADD, UPDATE, ARCHIVE, or SKIP
$\mathcal{E}^{(k+1)} \leftarrow \mathrm{ApplyEdits}(\bar{\mathcal{E}}, \Delta)$
$\mathcal{E}^{(k+1)} \leftarrow \mathrm{Compact}(\mathcal{E}^{(k+1)})$
return $\mathcal{E}^{(k+1)}$

Consolidation trigger. During GRPO training, consolidation is explicitly decoupled from parallel rollout generation. All workers use the same Experience snapshot within an epoch, and completed episodes and notes are buffered without modifying the persistent store. At the end of the rollout epoch, the collected evidence is gathered and consolidation is triggered once over the complete batch. The updated Experience store is then used in the following epoch. This keeps the external knowledge fixed within each rollout batch and avoids introducing additional non-stationarity into policy optimization.

Evolver operations. The evolver first abstracts accumulated trajectories into candidate Experience at two levels. Task-specific consolidation extracts reusable procedures and recurring failure patterns within a task family, while general consolidation identifies principles that transfer across task families. Each candidate is then reconciled with the current store. ADD introduces genuinely new Experience; UPDATE refines or corrects an existing entry; ARCHIVE removes stale, invalidated, or consistently unhelpful Experience from active retrieval while retaining it for provenance; and SKIP ignores evidence that is redundant or overly instance-specific. Capacity constraints and usage-based pruning further keep the active store compact.

Algorithm 1 summarizes the training-time procedure. Each worker independently collects trajectory evidence using the same frozen Experience snapshot. The worker-local buffers are gathered only after rollout collection finishes, after which the evolver performs a single consolidation step. Therefore, no worker observes an Experience update generated by another worker within the same rollout epoch.

I.3 POLICY-FACING BPE EXAMPLES

The following examples show representative mid-trajectory BPE views in the field structure used by our environment adapters. They are not complete agent prompts. EVOHARNESS-BASE receives these compact views automatically at each step. In EVOHARNESS-SFT and EVOHARNESS-RL, the same content becomes visible only after the corresponding track, commit, or recall action; the boxes combine these returned views to illustrate the external state available to subsequent decisions. Across all three environments, Experience follows the same two-level organization into general principles and task-specific procedures or failure patterns. Unlike a summary of the latest observation, these views retain facts, action outcomes, constraints, and unfinished requirements accumulated over a long interaction.

ALFWorld example. Consider a task that requires placing two tissue boxes on a dining table. After the agent has searched several receptacles, placed the first box, and picked up the second, the external state can expose:

Belief (episode-local)
Access: returned by track `[tissuebox]`.
Last successful action: pick up `tissuebox 2` from `coffeetable 1`.
Inventory: `tissuebox 2`.
Relations retained from earlier observations:
`tissuebox 1 on diningtable 1`
`tissuebox 2 held_by agent`
`tissuebox 3 on coffeetable 1`
Searched receptacles: `diningtable 1`, `coffeetable 1`.

Progress (episode-local)
✓ Find and place the first tissue box on the dining table.
✓ Locate and pick up the second tissue box.
□ Place the held tissue box on `diningtable 1`; then verify that two distinct boxes are on the target receptacle.

Experience (cross-episode)
General: Track repeated entities separately and verify each state transition before advancing a multi-object task.
Task-specific: Tissue boxes have frequently appeared on coffee tables; after each placement, verify the new relation before searching for the next instance. Do not treat the first placement as completion of a two-object goal.

The immediate observation need not repeat where the first box was placed or which receptacles were already searched. Belief preserves those facts, Progress keeps the remaining cardinality constraint explicit, and Experience supplies a reusable procedure without replacing instance-specific evidence. If a recalled location prior is contradicted, note records the correction for later consolidation rather than overwriting the current bank online.

WebShop example. For a shopping task, the agent may inspect several result pages and product variants before purchasing. The accumulated BPE state can render:

Belief (episode-local)
Access: assembled from targeted track calls.
Request constraints: navy; long sleeve; button-down; size L; price $< \$40$.
Compared candidates retained across search pages:
`B09A`: \$29.99, navy button-down, long sleeve, size L available.
`B07C`: \$35.50, navy button-down, short sleeve—reject.
Current item: `B09A`; selected options: navy, L; displayed price: \$31.99 after variant selection (within budget).

Progress (episode-local)
✓ Match color, sleeve style, and product type.
✓ Select the requested size and re-check the variant price.
□ Verify the final option summary and availability before purchase.

Experience (cross-episode)
General: Treat every requested attribute as a hard constraint and verify all selected options immediately before an irreversible action.
Task-specific: Search-result prices may differ from the price of the selected color/size variant; compare the post-selection price against the budget before *Buy Now*.

Belief functions as an external comparison table over pages that are no longer visible, while Progress prevents a plausible candidate from being purchased before every hard constraint is checked. Experience contributes only the transferable verification procedure, keeping it distinct from the current product facts.

WebArena example. For a multi-page task requesting monthly successful-order counts from an admin report, the policy may receive the following view after navigating through the dashboard and configuring several controls:

Belief (episode-local)
Access: rendered automatically in EVOHARNESS-BASE and exposed through BPE meta-actions in EVOHARNESS-SFT/RL.
Current page: Orders Report.
Navigation trail: Dashboard → Sales → Reports → Orders Report.
Active controls: From = 01/2023; To = 05/2023; Period = Month.
Recent outcomes: *Show Report* produced an error; repeated clicks on node `16130` failed because that element is no longer present in the current accessibility tree.

Progress (episode-local)
✓ Reach the sales report and set the date range and monthly period.
☐ Refresh the accessibility tree and submit the configured report.
☐ Collect five monthly counts and return them in `MM:COUNT` format.

Experience (cross-episode)
General: After a state-changing action or failed submission, re-observe the interface rather than reusing stale action references.
Task-specific: In admin reports, replace existing filter values and verify the rendered date and period controls before submitting; do not append to an old value.

Here, the current page alone does not explain how the agent arrived there, which filters were successfully set, or why the last action failed. Belief externalizes that execution history, Progress preserves the output contract, and Experience proposes a recovery rule grounded in the current failure. Thus, the adapter changes the concrete representation and update logic while preserving the same policy-facing BPE semantics.

J BASELINE DETAILS

Implementation and reporting protocol. For baselines evaluated by us, we adapt the authors' official implementations to our common environment interfaces and evaluation budgets whenever an official implementation is available; otherwise, we reproduce the method from the algorithm and prompts in the corresponding paper. Because SkillOS adopts the same evaluation setting and Qwen3-8B backbone as ours, the ALFWorld and WebShop results for ReasoningBank, MemP, SkillOS-base, and SkillOS in Table 1 are transcribed from the SkillOS paper (Ouyang et al., 2026). Results on WebArena and all remaining entries are from our evaluations unless stated otherwise. For these transcribed rows, we recompute ALFWorld Avg. from the reported per-category means using our 140-task category composition.

ReAct. ReAct (Yao et al., 2022b) interleaves free-form reasoning traces with executable environment actions. After every action, the resulting observation is added to the trajectory and conditions the next thought–action decision. It does not maintain cross-episode memory or update model parameters, and therefore serves as the memory-free agent backbone for both the frontier and open-source comparisons.

ExpeL. ExpeL (Zhao et al., 2024) learns from a pool of self-generated experience without gradient updates. It first collects successful and failed trajectories through trial, reflection, and retry; then contrasts failure–success pairs and groups of successful runs to induce reusable natural-language insights. The insight store is maintained through ADD, EDIT, UPVOTE, and DOWNVOTE operations. At evaluation time, the agent is prompted with these global insights together with retrieved successful trajectories similar to the current task.

ReasoningBank. ReasoningBank (Ouyang et al., 2025) distills both successful and failed trajectories into structured memory items containing a title, a short description, and reusable reasoning content. An LLM judge supplies a proxy success signal: successes contribute validated strategies, whereas failures contribute counterfactual lessons and pitfalls. For a new task, embedding similarity retrieves the most relevant items and injects them into the agent's instructions; newly extracted items are subsequently consolidated into the bank. We use the ReasoningBank memory loop, not its optional memory-aware test-time-scaling extension.

MemP. MemP (Fang et al., 2026) models procedural memory through three stages: building memory from completed trajectories, retrieving task-relevant memory, and updating the store from new

execution feedback. Its proceduralization representation combines concrete trajectories with abstract scripts, so the agent receives both an executable example and a general procedure. Retrieval uses semantic task keys, while online management can append new procedures, retain only validated successes, or revise a retrieved procedure when it leads to failure.

Dynamic Cheatsheet. Dynamic Cheatsheet (Suzgun et al., 2026) equips a frozen black-box model with an evolving non-parametric memory. A generator answers each query using the current cheatsheet, and a separately prompted curator judges the generated solution and adds, revises, merges, or removes compact strategies, solution sketches, and code snippets. Its retrieval-and-synthesis variant additionally retrieves similar past input–output pairs before curation. Thus, adaptation occurs entirely in context and does not require labels or parameter updates.

ACE. ACE (Zhang et al., 2025a) treats context as an evolving, structured playbook. A generator produces task trajectories, a reflector extracts lessons from successes and errors, and a curator converts those lessons into localized delta updates. Playbook entries are itemized bullets with usefulness and harmfulness statistics; new bullets are appended, existing ones are updated in place, and semantic deduplication periodically removes redundancy. This grow-and-refine design preserves prior knowledge while avoiding repeated monolithic prompt rewrites.

GRPO. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is the memory-free training baseline. For each task, it samples a group of trajectories and standardizes their rewards within the group to obtain relative advantages, thereby avoiding a separately trained critic. The policy is optimized with a PPO-style clipped surrogate and KL regularization against a reference policy. Unlike the skill-based methods below, vanilla GRPO neither retrieves nor updates an external experience store.

SkillOS. SkillOS (Ouyang et al., 2026) uses a frozen executor that retrieves relevant `SKILL.md`-style files from an external SkillRepo with BM25 and a separate curator that issues structured INSERT, UPDATE, or DELETE operations after each task. SkillOS trains this curator with GRPO on streams of related tasks: experiences from early tasks update SkillRepo, and performance on later related tasks measures whether those curation decisions were useful. Its composite reward combines downstream task outcomes with operation-validity, skill-quality, and conciseness signals. SkillOS-base is the corresponding variant that uses the initial curator without RL training, thereby isolating the effect of the architecture and prompting from learned skill curation.

SkillRL. SkillRL (Xia et al., 2026) jointly improves a skill-augmented policy and its external SkillBank. A teacher first distills successful trajectories into reusable strategies and failed trajectories into concise counterfactual lessons, then organizes them into always-available general skills and semantically retrieved task-specific skills. Skill-augmented teacher traces provide a cold-start SFT stage, after which GRPO optimizes the agent policy. Following validation epochs, failure patterns trigger additions or revisions to SkillBank, so the policy and hierarchical skill library evolve recursively. Because access to the original teacher model is constrained by API availability, we initialize SkillRL with the warm-up SkillBank released by its authors and use Claude Opus 4.6 as the teacher during online skill evolution.

K Implementation Details

This section records the implementation choices used on ALFWorld, WebShop, and WebArena. We follow the shared BPE interface and cost-aware objective in Section 2; Table 10 highlights the benchmark-specific settings that are most relevant for reproduction.

Model roles. We use Qwen3-8B and Qwen3-4B as trainable policy backbones; the headline results use Qwen3-8B, while Qwen3-4B is included in the cross-backbone study. Only the policy is updated by SFT and GRPO. We use Claude Opus 4.6 to collect successful SFT trajectories. To align the comparison with SkillRL, whose pipeline also uses a strong teacher model for skill construction and evolution, the runs reported in Table 1 use Claude Opus 4.6 as the Experience consolidator. The consolidator operates only between rollout batches, never inside an environment step, and

does not replace the Qwen3-8B policy that interacts with the environment. We further test whether our method depends on a frontier consolidation model. In the Self-Consolidate variant of Table 4, Qwen3-8B itself performs consolidation without assistance from any frontier model. This variant reaches 94.3% on seen tasks and 88.1% on unseen tasks, still outperforming all baselines while remaining close to the 95.0% and 94.8% obtained with Opus-based consolidation. These results reveal that the consolidation-model gap is concentrated on unseen tasks. We hypothesize that Qwen3-8B tends to extract more direct and homogeneous Experience tied to observed task patterns, whereas the frontier model more readily abstracts transferable principles. This distinction is expected because Qwen3-8B has not been specifically trained for Experience consolidation. Yet its strong self-consolidation performance shows that the effectiveness of our paradigm does not come merely from outsourcing reasoning to a frontier model: even an uncustomized 8B policy can construct a useful evolving Experience store, while a stronger consolidator primarily improves the generality of the knowledge it produces.

Action parsing and execution. The environment adapters parse each response into either an environment action or one of track, commit, recall, and note. ALFWorld and WebShop use the `<think>...</think><action>...</action>` format, while WebArena follows its native fenced-action format. Malformed or out-of-space actions receive failure feedback. Fixed penalties for invalid and degenerate repeated actions are included in the base reward R_{GRPO} of Equation 5, rather than introduced as additional shaping terms. We disable the trainer's built-in invalid-action penalty to avoid counting the same failure twice.

Harness grounding. All harness state is constructed from agent-visible observations and action outcomes. ALFWorld uses a rule-based object-state and location tracker; WebShop stores request constraints and product information; and WebArena maintains a compact application state from the accessibility tree and interaction history. Progress and Belief reset each episode, while Experience persists across episodes. Each Experience category has a maximum capacity of 80 entries; this limit applies separately to every task-family-specific category rather than to their aggregate. Note outputs are buffered for later consolidation. Evaluation uses two Experience protocols. For the training-based EvoHarness-RL, we freeze the Experience bank saved at the selected training checkpoint: evaluation trajectories may retrieve from this bank, but buffered notes are discarded and no consolidation is performed. For the frontier-model EvoHarness-Base results, the Experience bank starts empty and evolves across episodes on the evaluation split through the same batched consolidation mechanism between evaluation batches; the model parameters remain frozen. Experience is updated only from completed preceding evaluation episodes; no information from future episodes or unrevealed task instances is used, consistent with the other frozen self-evolving baselines. Thus, the former measures generalization with training-acquired Experience, whereas the latter measures an inference-only scaffold with online Experience evolution.

SFT data construction. For each benchmark, the teacher acts with the same BPE interface used by the policy, and we retain only successful episodes before flattening trajectories into next-action examples. This yields 87 trajectories and 1,153 next-action conversation pairs on ALFWorld, 423 trajectories on WebShop, and 187 trajectories and 1,327 next-action conversation pairs on WebArena. SFT uses LoRA with rank 64 and scaling 128. We optimize with AdamW, a cosine schedule, 10% warmup, and gradient clipping at 1.0. The resulting merged checkpoint initializes full-parameter GRPO.

GRPO training. GRPO samples 128 trajectories per update on every benchmark, normalizes rewards within each prompt group, and regularizes the policy against the SFT reference. We use PPO clipping 0.2, entropy coefficient 0.001, weight decay 0.01, and gradient clipping 1.0. Diversity shaping is cosine-annealed over training. For rollout construction, we follow the interactive setup of SkillRL (Xia et al., 2026) for ALFWorld and WebShop, with prompt and response token limits applied per generation turn. For WebArena, we follow AgentGym-RL's ScalingInter setup (Xi et al., 2025), which serializes multi-turn interactions into trajectory-level sequences for optimization. All three environments use on-policy GRPO rollouts. ALFWorld training episodes allow up to 70 steps, providing additional interaction budget for early exploration of the newly introduced meta-actions; WebShop training episodes allow up to 20 steps. For a consistent comparison, all trainable WebArena methods use the AgentGym-RL setup, running for 300 updates with an interaction-horizon

Setting	ALFWorld	WebShop	WebArena
Model and SFT			
Policy backbones	Qwen3-8B and Qwen3-4B		
SFT teacher	Claude Opus 4.6		
LoRA rank r and scaling α	64 and 128		
SFT learning rate	1×10^{-5}		
SFT epochs	4	4	3
GRPO			
Group size G	8	8	4
Prompts per step	16	16	32
Trajectories per update	128		
PPO mini-batch	128	32	4
PPO inner epochs	1	1	2
Training updates	150	150	300
Training interaction horizon	70	20	$8 \rightarrow 10 \rightarrow 15$
Actor learning rate	1×10^{-6}		
KL coefficient	0.01		
Maximum prompt tokens	12,288	8,192	2,048
Maximum response tokens	512	1,024	8,192[†]
vLLM tensor parallelism	4	2	1
RL hardware	8×H200	8×H200	16×H200
Harness, evaluation, and reward			
Maximum evaluation steps	50	20	15
Recall top-k_{rec}	3	6	3
Maximum Experience entries per category	80		
Success scale in R_{GRPO}	10.0	10.0	1.0
Efficiency weight λ_{eff}	1.0	1.0	0.02
Max diversity weight $\lambda_{\text{div}}^{\max}$	1.0	1.0	0.02
Repeated-action penalty in R_{GRPO}	0.1	0.1	0.01
Invalid-action penalty in R_{GRPO}	0.1	0.1	0.01

Table 10: Implementation and training hyperparameters for EVOHARNESS-RL across the three benchmarks. Shared optimizer defaults are given in the text. [†]WebArena limits each generation turn to 512 tokens and the complete trajectory response to 8,192 tokens.

curriculum of 8, 10, and 15 turns. At evaluation, all methods share the same interaction budgets: 50 steps on ALFWorld, 20 on WebShop, and 15 on WebArena. Both environment actions and harness operations count toward these budgets.

L PROMPTS

We reproduce the key prompts that instantiate the BPE harness on ALFWorld, WebShop, and WebArena. EVOHARNESS-BASE receives automatically rendered BPE views, whereas the adaptive prompts used for EVOHARNESS-SFT and EVOHARNESS-RL expose the same four meta-actions with environment-specific action semantics and no prescribed invocation cadence. The consolidation prompts operate outside the rollout loop and turn trajectory evidence into structured Experience updates. WebArena retains its native browser-agent conversation and action format. The prompt text is lightly abridged for readability.

L.1 RUNTIME HARNESS PROMPTS

Shared structure. Under EVOHARNESS-BASE, each environment adapter automatically appends the current BPE view to the policy context without requiring an additional action. The adaptive prompts used for EVOHARNESS-SFT and EVOHARNESS-RL instead define the environment actions, expose $\mathcal{A}_{\text{bpe}} = \{\textbf{commit}, \textbf{track}, \textbf{recall}, \textbf{note}\}$, and specify the response syntax. They describe action semantics without prescribing an invocation order or cadence. Across environments,

commit externalizes the current subgoal, **track** queries episode-local Belief, **recall** retrieves cross-episode Experience, and **note** records reusable evidence. The environment-specific text changes what can be tracked and recalled: object locations in ALFWorld, product constraints in WebShop, and previously observed pages and values in WebArena.

L.1.1 ALFWORLD

The ALFWorld prompt grounds the four actions in household manipulation.

Harness system prompt (policy-facing)

You are an autonomous intelligent agent operating in the ALFWorld text-based household environment. The objective is task completion through interactions with objects in rooms.

You will receive: OBJECTIVE, OBSERVATION, ADMISSIBLE COMMANDS, PREVIOUS ACTION(S).

Environment Actions (listed in ADMISSIBLE COMMANDS)
- go to <recep>, take <obj> from <recep>, put <obj> in/on <recep>
- open/close <recep>, heat/cool/clean <obj> with <appliance>, use <tool>
- examine/look/inventory

Harness Actions (cognitive tools; each call costs one step)
Plan
- **commit** [subgoal]: Register the current subgoal. e.g. **commit** [find egg]
Perception
- **track** [object]: Query a specific object's last location, state, and visited locations. e.g. **track** [egg]
Experience
- **recall** [query]: Retrieve past experience. What you get depends on the query:
  recall [where to find X] → search hints (object→location)
  recall [how to do Y task] → procedures, general skills, common mistakes
  recall [mistakes to avoid] → common pitfalls and fixes
- **note** [insight]: Record a generalizable discovery for future episodes. e.g. **note** [food usually in fridge or countertop]

Response Syntax
<think>Brief reasoning in 1-2 sentences.</think>
<action>your action here</action>

Each response contains one action. Environment-action strings come from ADMISSIBLE COMMANDS.

At each step the policy additionally receives a per-turn user message carrying the current context x_t and the rendered harness views h_t (the committed plan, the last recalled hints, and any **track**/**recall** results):

Per-turn user template

OBJECTIVE: {task_description}
OBSERVATION: {current_observation}
ADMISSIBLE COMMANDS: {admissible_commands}
PREVIOUS ACTION(S): {action_history}
{harness_views} # PLAN, RECALLED HINTS, track/recall results

L.1.2 WEBSHOP

The WebShop prompt preserves the same interface but grounds Belief in the shopping state: price, candidate products, and selected options. The block below is the RL policy prompt.

WebShop harness prompt (RL policy)

You are an autonomous intelligent agent operating in the WebShop e-commerce environment. The objective is to buy the product that best matches the instruction, including its options and price ceiling.

You will receive: OBJECTIVE, OBSERVATION, AVAILABLE ACTIONS, PREVIOUS ACTION(S).

Environment Actions (listed in AVAILABLE ACTIONS)
- `search[<query>]` on the search page.
- `click[<button>]` for a product, option, navigation control, or `Buy Now`.

Harness Actions (each call costs one step)
- **commit** [subgoal]: register the current shopping subgoal.
- **track** [price]: compare the current price with the budget.
- **track** [candidates]: list previously seen products and prices.
- **track** [options]: show selected options on the current item.
- **recall** [query]: retrieve effective searches, purchase procedures, or mistakes to avoid.
- **note** [insight]: save a reusable search phrasing or a shopping pitfall.

Response Syntax
<think>Brief reasoning in 1–2 sentences.</think>
<action>your action here</action>
Each response contains one action. Environment-action strings come from AVAILABLE ACTIONS.

During SFT, WebShop uses the same action names and semantics.

WebShop per-turn user template

OBJECTIVE: {task_description}
OBSERVATION: {current_observation}
AVAILABLE ACTIONS: {available_actions}
PREVIOUS ACTION(S): {action_history}
{harness_views}

L.1.3 WEBARENA

In EVOHARNESS-BASE, WebArena automatically appends the current compact BPE view to the native browser-agent context, and the policy emits only browser actions. The prompt below instantiates the adaptive interface used by EVOHARNESS-SFT and EVOHARNESS-RL. The policy observes an accessibility tree, URL, open tabs, and prior actions and explicitly chooses between browser and BPE meta-actions. The full conversation is retained across rounds, and both action types are emitted in triple-backtick blocks rather than `<action>` tags.

WebArena browser and cognitive-tools prompt

You are an autonomous intelligent agent tasked with navigating a web browser. You receive the user's objective, the current page's accessibility tree and URL, the open tabs, and the previous action.

Browser Actions
- Page operations: `click [id]`, `type [id] [content] [0|1]`, `hover [id]`, `press [key]`, and `scroll [down|up]`.
- Navigation: `new_tab`, `tab_focus [i]`, `close_tab`, `goto [url]`, `go_back`, and `go_forward`.
- Completion: `stop [answer]`.

Cognitive Tools (page-independent; each call consumes one interaction step)
- **commit** [subgoal]: declare the goal you are about to pursue, not a result.
- **track** [visited]: list previously opened pages and URLs.
- **track** [values]: recover numbers seen on earlier pages.
- **track** [objective]: recover earlier text related to the objective; any other argument is matched literally against previously observed page text.
- **recall** [how to do <task type>]: retrieve procedures and principles.
- **recall** [mistakes to avoid]: retrieve common pitfalls.
- **note** [insight]: save a reusable procedure, feature location, or pitfall; its content represents reusable experience rather than the current task's answer.

Response Syntax
A response consists of step-by-step reasoning followed by one browser or cognitive-tool action inside a triple-backtick block. `stop [answer]` represents task completion.

WebArena conversation layout

SYSTEM: You are Qwen, created by Alibaba Cloud. You are a helpful assistant.
USER: {browser instruction + cognitive-tools block}
ASSISTANT: Ok.
USER: {round-1 observation}
ASSISTANT: {round-1 reasoning and action}
... the full user/assistant history is retained

L.2 Experience-Store Consolidation Prompts

Experience evolution uses three prompt blocks: task-specific abstraction, general abstraction, and reconciliation. The main experiments initialize the Experience bank as empty; teacher trajectories supervise use of the BPE interface while the bank used by the main RL runs remains unseeded. During RL, the bank remains frozen within each rollout batch and is updated only between batches. The two abstractors distill evidence from trajectories and policy-written notes, after which the reconciler folds supported candidates into the current bank.

Shared system message. All three calls use the same system message. Environment-specific slots retain their benchmark-specific bindings.

Shared consolidation system message

You are a helpful assistant. Reasoning may precede the FINAL output. The FINAL output format is a single valid JSON value, with no prose or markdown fences.

Task-specific abstraction. This prompt is applied to each task type represented in the rollout batch.

Task-specific Experience abstraction

You are the task-experience abstractor for {agent_description}. You are given the episodes produced for one {task_specific_noun}. The output is candidate Experience distilled from this evidence alone; the store remains unchanged.
{task_label}: {task_type}
SUCCESSFUL EPISODES: {success_data}
FAILED EPISODES: {failure_data}
AGENT NOTES: {notes_text}
Return a JSON object containing candidate skills, mistakes, and search priorities. Each candidate contains a unique ID, a short actionable principle, its trigger condition, and supporting evidence. Candidates are specific to {task_type}; a compact set of high-value entries is preferred, with empty lists representing a lack of supported candidates.

General abstraction. This prompt operates once over the pooled batch.

General Experience abstraction

You are the general-experience abstractor for {agent_description}. You are given episodes produced across several {task_specific_noun}s. The output is candidate Experience that transfers across them; the store remains unchanged.
{task_label}S PRESENT IN THIS BATCH: {task_type_list}
SUCCESSFUL EPISODES: {success_data}
FAILED EPISODES: {failure_data}
AGENT NOTES: {notes_text}
Return a JSON object containing general candidate skills and mistakes. Retain a candidate only when at least two task types support it, name those task types, and cite the supporting trajectories or notes. Omit

task-specific observations and search priorities. Prefer a few short, high-value candidates, and return empty lists when appropriate.

Reconciliation. Task-specific and general candidates are reconciled with the frozen bank.

Experience reconciliation

You are the reconciler for {agent_description}'s experience store. You are given candidates distilled from the latest rollout batch and the frozen store. The output represents incremental edits that fold supported candidates into the store rather than restating the full store.

STAGE: {stage}
{stage_rules}
CANDIDATE SKILLS: {candidate_skills}
CANDIDATE MISTAKES: {candidate_mistakes}
CANDIDATE SEARCH PRIORITIES: {candidate_search_priorities}
EXISTING SKILLS: {existing_skills}
EXISTING MISTAKES: {existing_mistakes}

Return a JSON object containing Experience operations and supported search priorities. Each candidate maps to one of ADD, UPDATE, ARCHIVE, or SKIP. SKIP represents a redundant candidate, UPDATE a refinement or correction, ARCHIVE an entry that is misleading beyond repair, and ADD genuinely new Experience. Update and archive operations reference existing IDs; evidence references come from the provided inputs, and fields are short.

Global reconciliation edits general Experience, while local reconciliation may also edit the current task type and its search priorities. The implementation may fuse abstraction and reconciliation into a single batched call, while preserving the same ADD, UPDATE, ARCHIVE, and SKIP semantics. ARCHIVE removes an entry from active retrieval while retaining it for provenance; applying edits and enforcing the capacity limit are deterministic operations.