RESPONSIVEENV: EVOLVING TRAINING WORLDS FOR LLM AGENTS THROUGH ONLINE EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents learn by interacting with their environments. Recent work on environment scaling seeks to improve agent learning by expanding the number and diversity of training environments. We instead obtain more useful experience from existing worlds by revising their conditions while preserving task intent. A limited pool of original worlds can then provide varied practice for the same tasks. Useful conditions depend on the learner's current capabilities. We therefore propose **`ResponsiveEnv`**, a framework that revises world conditions throughout training based on current learner trajectories. An environment editor simplifies or hardens selected worlds, and validated revisions replace them for subsequent training. We evaluate **`ResponsiveEnv`** on unedited ALFWorld and τ^2-bench tasks. On ALFWorld, we edit 100 original worlds with Claude Opus 4.8, GPT-5.6 Sol, or a distilled 8B editor. Each editor raises the unseen-task success rate by 18 to 19 percentage points over static training on the same worlds. Editing these 100 worlds also outperforms training on 2,288 static worlds under the same learner-update budget. For the 8B editor, replacing the current learner's trajectories with success rates alone, or with the initial policy's trajectories, lowers the unseen-task success rate by 6.3 and 5.5 points, respectively.

1 INTRODUCTION

Training environments determine which situations LLM agents practice, such as searching for objects or sequencing tool calls, and thus which skills they acquire (Wang et al., 2024; Xi et al., 2025; Li et al., 2026). Reinforcement learning from interaction is now a standard approach for training such agents (Silver & Sutton, 2025; Zhang et al., 2026; Wang et al., 2025; Xi et al., 2026; Feng et al., 2025). To supply more interaction, recent work expands training experience (Huang et al., 2025; Li et al., 2026) through environment and task synthesis (Hu et al., 2025; Song et al., 2026), executable tool environments (Wang et al., 2026b), and simulated interaction feedback (Chen et al., 2026). These approaches expose agents to more diverse interactions during training, and such diversity supports generalization (Cobbe et al., 2020).

The value of training experience also depends on the learner's current capabilities (Bengio et al., 2009; Narvekar et al., 2020). Automatic curricula address this dependence by adapting which tasks and environments an agent encounters (Florensa et al., 2017; Dennis et al., 2020; Portelas et al., 2020; Jiang et al., 2021; Parker-Holder et al., 2022). For LLM agents, recent work uses learner feedback to generate tasks and adapt training environments (Qi et al., 2025; Huang et al., 2026; Dong et al., 2026). These methods mostly add new tasks or environments, or build a fixed set of variants once. The conditions of existing tasks are rarely revised again as the learner changes. We study this opportunity: repeatedly revising the conditions under which existing tasks are practiced. Object locations, distractors, or initial conditions can change the decisions a task requires while its intent stays fixed. Whereas domain randomization samples such conditions from a fixed distribution (Tobin et al., 2017), we adapt them to the current learner. We ask two questions: *Can repeatedly revising existing worlds under task constraints improve agent learning from a limited pool of original environments? What makes such revisions useful?*

For example, an agent may fail a multi-step task because it cannot find a required object or piece of information. Making that resource easier to locate gives the agent more opportunities to practice the remaining steps. As the agent learns to find the resource, adding plausible distractors makes the same

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.

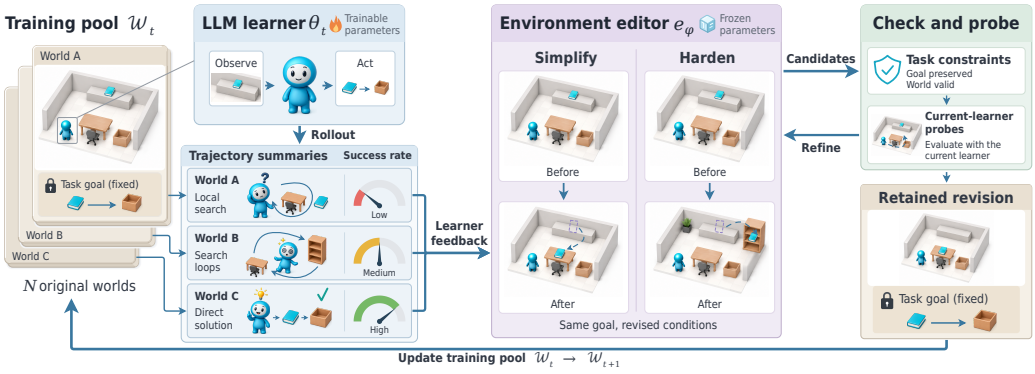

Figure 1: **Evolving training worlds with `ResponsiveEnv`.** Learner success rates and trajectory summaries guide a frozen editor to simplify or harden world conditions while preserving task goals. Task checks and learner probes guide refinement. Retained revisions replace previous worlds in the training pool, where the learner continues training and generates fresh feedback for subsequent edits. Rooms schematically illustrate world states in text-based tasks.

task more demanding without changing its objective. Which change helps depends on the agent's current behavior: relocating an object matters only relative to where the agent now searches.

This suggests a way to keep existing worlds useful as training progresses: adapt their conditions to the learner's current capabilities. We implement this idea by repeatedly editing executable worlds in response to current learner feedback. We call this online editing framework **`ResponsiveEnv`**.

As in recent LLM-guided environment design (Zala et al., 2024; Liang et al., 2025; Guo et al., 2025; Liu et al., 2026), an LLM serves as the environment editor. It receives the current world, its task, and a summary of learner trajectories and outcomes. It then proposes structured changes that simplify or harden world conditions. Success rates decide which worlds to revise and in which direction, while trajectory summaries inform what to change. Domain-specific validators enforce task constraints. In ALFWorld (Shridhar et al., 2021), these checks require the task goal to remain unchanged and use a planner to verify solvability. We evaluate valid edited worlds with the current learner, using a target success rate to guide selection. The selected edited worlds replace their previous versions in the training pool and become the starting points for later revisions. The editor's parameters remain fixed throughout learner training.

We evaluate Qwen3-8B learners (Yang et al., 2025) through complete interactions on ALFWorld and τ^2-bench tasks (Barres et al., 2025). We study external editors, including Claude Opus 4.8 (Opus) and GPT-5.6 Sol (GPT), alongside distilled local editors. Our coverage experiments measure *environment efficiency*: held-out performance relative to the number of original training worlds at a fixed learner-update budget. On ALFWorld, editing 100 original worlds reaches an unseen-task success rate of 86.15% with Opus and 85.77% with a distilled 8B editor, while static training reaches 67.21% on the same worlds and 82.74% on 2,288 worlds. At the same number of learner updates, 100 edited worlds thus outperform 2,288 static ones. On τ^2-bench, external and distilled editors provide complementary gains in telecom and retail.

Our contributions are as follows:

- We develop an online world-editing framework that uses learner feedback to simplify or harden existing worlds under task constraints. Selected edited worlds are retained for subsequent training and revision.
- We show that editing 100 original ALFWorld worlds reaches higher held-out success than training on 2,288 static worlds under the same learner-update budget. This holds with Opus, GPT, and a distilled 8B editor.
- We show that the gains require repeated revision conditioned on the current learner. An easy-to-hard curriculum, one-time editing, random edit operations, and success-only feedback each leave unseen-task success 5 to 8 points below the full method. Behavioral summaries help only when they come from the current learner, and a case study shows one world simplified and later hardened as the learner improves.

2 RESPONSIVEENV

`ResponsiveEnv` interleaves agent training with edits to existing worlds while preserving task intent (Figure 1). At each editing round, an environment editor uses current learner feedback to choose changes to world conditions. Success estimates guide which worlds to revisit and whether to simplify or harden them, while trajectory summaries inform the specific edits. Candidate revisions are checked against task constraints and evaluated with the current learner. Selected revisions replace the previous worlds in the training pool, supporting subsequent training and serving as the starting points for later edits.

2.1 TASK-PRESERVING WORLD EDITING

We represent each task instance by a goal g and a world configuration w. The goal specifies the desired outcome, while the world configuration describes the state of the environment, including the properties and relationships of its entities. Revising this configuration changes the situations the agent encounters and the interactions needed to complete the same task.

At editing round t, we maintain a training pool of N worlds indexed by i, $\mathcal{W}_t = \{(w_{i,t}, g_i)\}_{i=1}^{N}$. An edit a specifies an operation and its arguments. We write $w' = f_{\text{apply}}(w, a)$, where f_{apply} executes a on w through the environment's editing interface and returns the candidate configuration w'. A domain-specific validator $C_d(w, w', g) \in \{0, 1\}$ checks whether w' satisfies the domain's task-preservation contract.

Edits may change entities, their attributes, and their relationships while preserving the task goal, action semantics, and feasibility. The required interactions and solution length may vary within these constraints (Appendices A.3 and A.4).

2.2 LEARNER-CONDITIONED WORLD EDITING

We use current learner feedback to choose edits within these task constraints. Success rates indicate how difficult a world is for the learner; trajectories show which conditions shape that difficulty. We therefore provide the editor with both: success estimates decide which worlds to revise and in which direction, and trajectory summaries inform what to change. Section 4.1 tests this division.

Let π_{θ_t} denote the learner at editing round t. Given a per-round editing budget M, the selection function f_{select} uses recent training outcomes and a target success band $\mathcal{B}$ to return at most M world indices for revision. Worlds with estimated success below the band are considered for simplification, while those above it are considered for hardening. We use $\mathcal{B} = [0.35, 0.65]$ and, in ALFWorld, edit up to $M = 25$ worlds every $H = 15$ learner updates. For each selected world, we collect fresh trajectories $\mathcal{T}_{i,t}$ from the current learner and construct the editor's input:

$$x_{i,t} = \big(w_{i,t}, g_i, \Sigma(\mathcal{T}_{i,t}), \widehat{p}_{\theta_t}(w_{i,t}, g_i)\big), \tag{1}$$

Here, $\widehat{p}$ is the fraction of successful rollouts, and Σ is a deterministic function that summarizes action usage, interaction patterns, and trajectory lengths. Behavioral summaries describe the successful rollouts, which show what the learner relies on; when no rollout succeeds, they describe the failed ones. The editor interprets these observations together with the world configuration and task goal to propose concrete changes. Their effects are then evaluated through interaction with the current learner, as described in Section 2.3.

2.3 VALIDATION AND LEARNER-GUIDED REFINEMENT

We use two editor interfaces that share the input packet in Eq. (1), validation, and retention. External editors (Opus, GPT, and Gemini) run interactive sessions, described below. Local editors, such as a distilled 8B editor (SFT-8B), use a single-response interface: they propose K edit lists for the same world, and valid candidates are ranked by fresh learner probes (Appendix A.3). The comparisons in Section 4.1 use this interface.

An interactive session starts from a copy of the current world. Its state z tracks the current candidate, saved worlds, and probe records. The editor can inspect objects and receptacles, modify world

Algorithm 1 `ResponsiveEnv`: online world editing with an interactive editor

Require: Original pool $\mathcal{W}_0 = \{(w_{i,0}, g_i)\}_{i=1}^{N}$, learner θ_0, fixed editor e_ϕ
Require: S segments of H learner updates, at most M worlds edited per round
Require: Band $\mathcal{B}$, target $p^\star$, validator C_d, and session budgets

$\mathcal{W} \leftarrow \mathcal{W}_0$; $\theta \leftarrow \theta_0$
for $s = 1, \dots, S$ **do**
  $(\theta, \mathcal{T}) \leftarrow f_{\text{train}}(\theta, \mathcal{W}, H)$ ▷ Section 2.4
  if $s < S$ **then**
    $\mathcal{I} \leftarrow f_{\text{select}}(\mathcal{W}, \mathcal{T}, M, \mathcal{B}, p^\star)$ ▷ Section 2.2
    for all $i \in \mathcal{I}$ **do**
      $(w, g) \leftarrow \mathcal{W}[i]$
      $(\widehat{p}, \mathcal{T}_i) \leftarrow f_{\text{probe}}(\pi_\theta, w, g)$ ▷ Fresh rollouts
      $x \leftarrow (w, g, \Sigma(\mathcal{T}_i), \widehat{p})$ ▷ Behavior and difficulty; Eq. 1
      Initialize session z from $(w, g, p^\star)$; $h \leftarrow \emptyset$
      while the session is open and interaction budget remains **do**
        $u \sim e_\phi(\cdot \mid x, h)$ ▷ Use current feedback to choose an action
        $(z, o) \leftarrow f_{\text{tool}}(z, u, \pi_\theta, C_d)$ ▷ Section 2.3
        Append (u, o) to h ▷ Use feedback for the next action
      end while
      $w' \leftarrow f_{\text{retain}}(z, w, g)$ ▷ Appendix A.3
      if $w' \neq$ `None` **and** $C_d(w, w', g) = 1$ **then**
        $\mathcal{W}[i] \leftarrow (w', g)$ ▷ Persist for training and later revision
        Record $f_{\text{probe}}(\pi_\theta, w', g)$ ▷ Independent seeds; diagnostic only
      end if
    end for
  end if
end for
return π_θ

conditions, and request learner rollouts on a candidate revision. It can also save and restore candidate states. The learner checkpoint stays fixed within the session, so all probes evaluate the same policy.

The probing function $f_{\text{probe}}(\pi_{\theta_t}, w', g)$ samples fresh learner rollouts on a candidate world and returns their empirical success rate and trajectories. These observations allow the editor to inspect how its changes affect learner behavior. It may then refine the candidate or restore an earlier state and try another revision. Its next tool action is $u_j \sim e_\phi(\cdot \mid x_{i,t}, h_j)$, where h_j is the history of tool calls and results. The execution function $f_{\text{tool}}(z, u_j, \pi_{\theta_t}, C_d)$ carries out this action, returning the updated session state and an observation o_j. Edit actions modify the candidate through f_{apply}. A successful commit closes the session; otherwise, search continues within the interaction budget.

Validation requires $C_d(w, w', g) = 1$, enforcing task constraints and solvability. To guide difficulty adjustment within this valid space, we use the heuristic

$$q(w; \theta) = -\left|\widehat{p}_\theta(w, g) - p^\star\right|, \qquad \Delta q(w, w'; \theta) = q(w'; \theta) - q(w; \theta), \tag{2}$$

with $p^\star = 0.5$. Interactive editors read probe outcomes against $p^\star$, and single-response ranking keeps the candidate with the highest q. The retention function $f_{\text{retain}}(z, w, g)$ returns a selected valid revision, or `None`, in which case the current world remains unchanged; Appendix A.3 motivates $p^\star$ and gives the commit and fallback rules. Separate post-edit rollouts, with seeds disjoint from candidate probes, measure each retained world's success; editing rollouts never enter learner batches.

The domain-specific τ^2-bench implementation is given in Appendix A.4. The prompts and tool interfaces are provided in Appendix G.

2.4 Learning with persistent world revisions

Retained revisions replace the corresponding worlds in the training pool. Let $\mathcal{R}_t$ denote the indices of worlds with a valid revision retained at round t. We update each world as

$$w_{i,t+1} = \begin{cases} w'_{i,t}, & i \in \mathcal{R}_t, \\ w_{i,t}, & \text{otherwise.} \end{cases} \tag{3}$$

Each entry preserves its task goal g_i, and the pool keeps one current version per original world.

The learner then trains on the updated pool $\mathcal{W}_{t+1}$. The function $f_{\text{train}}(\theta_t, \mathcal{W}_{t+1}, H)$ performs H learner updates and returns the updated parameters and training trajectories. We instantiate these updates with group-relative reinforcement learning (Shao et al., 2024). At the next editing round, the editor uses feedback from the updated learner to revise the latest world versions.

Algorithm 1 summarizes this alternating process. Appendix A.3 specifies the editing schedule and world selection variants.

3 Experiments

We evaluate whether revising existing training worlds improves agent performance on original tasks, how this compares with expanding the training pool, and how the choice and timing of edits matter.

3.1 Setup

Training details. We train Qwen3-8B with group-relative reinforcement learning (Shao et al., 2024) for 300 learner updates, with 64 trajectories per batch and a learning rate of 10^{-6}, using prime-rl (Prime Intellect, 2025). ALFWorld starts from the original model, while all τ^2-bench runs share a conversation-adapted checkpoint obtained by self-distillation.

Environment editors. On ALFWorld, we evaluate Opus, GPT, and Gemini 3.8 Flash (Gemini) as interactive editors. The τ^2-bench comparison uses Gemini and Opus as external editors. Our default local editor, *SFT-8B*, is Qwen3-8B fine-tuned on editing demonstrations from external editors to output a diagnosis and an edit list (Appendix A.3). All editors remain frozen during training.

Benchmarks and splits. We evaluate on ALFWorld (Shridhar et al., 2021) for text-based household tasks and τ^2-bench (Barres et al., 2025) for conversational tool use. All evaluations use original, unedited tasks. ALFWorld evaluation covers all six task families, with official Unseen (134 instances) and Seen (140) splits and an additional 400-instance Room-held-out set (Room) drawn from rooms excluded from learner training. The ALFWorld training pools contain 100, 500, or 2,288 original worlds within the same training-room boundary. For τ^2-bench, we use fixed test subsets of 40 telecom, 40 retail, and 20 airline tasks.

Baselines. Static baselines train on the original environments without editing. On ALFWorld, we adapt two environment-adaptation methods from the authors' released code (Appendix A.5). EnvHarness (Huang et al., 2026) uses an LLM designer to turn policy trajectories into episode-start actions and rule hooks. We build these components once before training, with Gemini 3.1 Flash-Lite (Flash-Lite) or GPT as the designer. RLAnything (Wang et al., 2026a) co-trains the policy, a reward model, and the environment; we use its environment step, in which frozen Qwen3-8B or GPT rewrites task goals. Each adaptation keeps its method's form of intervention and acceptance rule. Section 4.1 adds single-ingredient controls.

Metric. We report mean single-attempt success $\widehat{S}$ over 16 attempts per instance, averaging instances equally, and report one training run per configuration. Success rates are percentages and absolute differences $\Delta\widehat{S}$ are points. Figures and appendix tables report paired 95% confidence intervals (CIs) from instance-bootstrap resampling, conditional on the trained checkpoints. Appendix A gives full training, editor, and statistical settings; Appendices B and C report additional endpoints and comparison intervals.

3.2 Comparison with baselines

Table 1 compares complete training configurations starting from the same 100 original worlds. With Opus, **`ResponsiveEnv`** reaches 86.15% Unseen, 82.32% Seen, and 82.09% Room success, compared with 67.21%, 68.48%, and 69.98% for static environments (+18.94, +13.84, and +12.11 points). GPT obtains 85.35% Unseen success and the highest Room score among the 100-world configurations in the table. The distilled SFT-8B editor matches both, staying within 2 points of each on every split (85.77%, 80.71%, and 81.25%). With Gemini, a lighter model, Unseen success rises by 8.21 points to 75.42%: learner gains grow with editor capability, and distillation brings that capability to an 8B editor.

Table 1: ALFWorld success (%) after 300 learner updates from 100 original worlds. Bold and underlining mark the highest and second-highest scores in each column.

Method	Environment editor	Unseen	Seen	Room
Static environments	–	67.21	68.48	69.98
EnvHarness	Gemini 3.1 Flash-Lite	78.54	80.49	78.52
EnvHarness	GPT-5.6 Sol	69.22	67.77	71.31
RLAnything	Qwen3-8B	**86.57**	76.65	80.22
RLAnything	GPT-5.6 Sol	82.00	78.48	78.41
ResponsiveEnv	Gemini 3.8 Flash	75.42	72.41	75.27
ResponsiveEnv	GPT-5.6 Sol	85.35	80.40	**83.13**
ResponsiveEnv	Claude Opus 4.8	<u>86.15</u>	**82.32**	<u>82.09</u>
ResponsiveEnv	SFT-8B	85.77	<u>80.71</u>	81.25

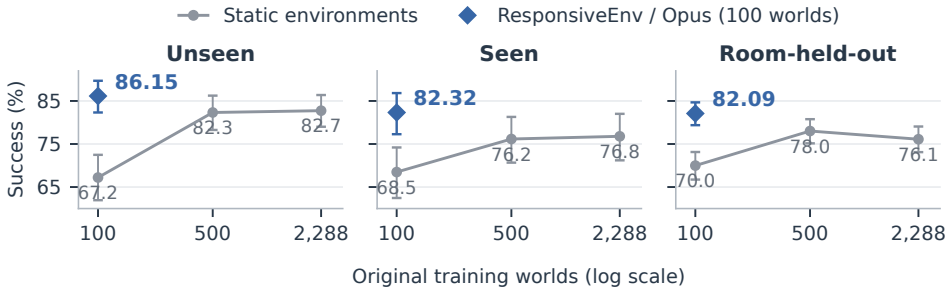

Figure 2: **Environment efficiency.** Static training with 100 to 2,288 original worlds versus editing 100 with Opus, evaluated after 300 updates. Bars show 95% instance-bootstrap CIs.

With GPT as the proposal model in both pipelines, **ResponsiveEnv** exceeds EnvHarness by 16.14, 12.63, and 11.81 points on Unseen, Seen, and Room, and the RLAnything protocol by 3.36, 1.92, and 4.72 points. Editor capability pays off through world editing: replacing each baseline's original proposer with GPT lowers its Unseen success (EnvHarness from 78.54% to 69.22%, RLAnything from 86.57% to 82.00%), whereas **ResponsiveEnv** gains about 10 points from Gemini to GPT. Against RLAnything's original Qwen3-8B configuration, Opus is on par on Unseen (86.15% versus 86.57%) and leads by 5.67 points on Seen.

3.3 Learning from fewer original environments

At a fixed learner-update budget, expanding the static ALFWorld pool from 100 to 500 worlds raises Unseen success from 67.21% to 82.32%, and using 2,288 worlds yields 82.74% (Figure 2): static training plateaus between 500 and 2,288 worlds. Editing 100 worlds with Opus reaches 86.15%, exceeding both expanded static pools across all three splits. Against the 2,288-world static pool, it gains 3.40, 5.54, and 5.98 points on Unseen, Seen, and Room, respectively. The GPT and SFT-8B editors also exceed the 2,288-world pool on all three splits (Table 1).

3.4 Conversational tool use

Table 2 reports success in each domain and the unweighted mean across domains. All three editors raise macro success over static training (40.16%): Opus to 44.01%, SFT-8B to 43.91%, and Gemini to 42.34%. Opus and SFT-8B lead in different domains. Opus gains most in telecom, from 50.47% to 57.50% (+7.03 points). SFT-8B, trained on Opus-generated editing demonstrations for these domains, gains most in retail, from 49.06% to 58.28% (+9.22 points), and has the highest airline score. Additional editor configurations and task-weighted aggregates appear in Table 16, with intervals in Appendix B.

Table 2: τ^2-bench success (%) on fixed test subsets after 300 learner updates. Macro averages domains equally. Bold and underlining mark the highest and second-highest scores in each column.

Method	Environment editor	Telecom	Retail	Airline	Macro
Static environments	–	50.47	49.06	20.94	40.16
`ResponsiveEnv`	Gemini 3.8 Flash	<u>52.03</u>	<u>53.44</u>	21.56	42.34
`ResponsiveEnv`	Claude Opus 4.8	**57.50**	51.72	<u>22.81</u>	**44.01**
`ResponsiveEnv`	SFT-8B	48.44	**58.28**	**25.00**	<u>43.91</u>

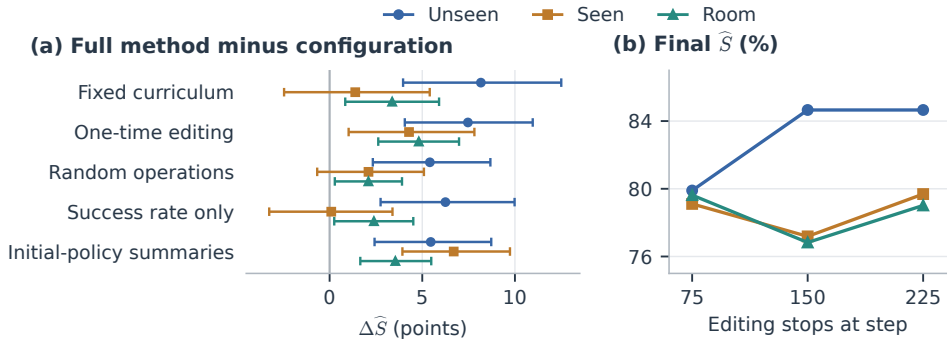

Figure 3: **Editing strategies and timing.** (a) Full method minus each configuration in Table 3, with 95% instance-bootstrap CIs. (b) Final success at update 300 after stopping edits at different steps.

4 Analysis of World Editing

4.1 What do the gains require?

Table 3 separates the ingredients of the full method, **`ResponsiveEnv`** with SFT-8B and current-learner feedback (85.77% Unseen; the last row), and Figure 3(a) gives its paired gap to each configuration; Table 17 in Appendix C defines the five controls. On Unseen, the full method leads the fixed curriculum by 8.16 points, one-time editing by 7.46 points, random operations by 5.41 points, success rate only by 6.25 points, and initial-policy summaries by 5.46 points; on Room, it leads all five by 2.09 to 4.81 points. Reordering the original worlds into a fixed easy-to-hard curriculum raises Unseen success from 67.21% to 77.61%, and editing them once before training with the same editor reaches 78.31%. Online revision helps most when proposals draw on the current learner's behavior: random operations, which keep current-learner world selection and candidate ranking, reach 80.36%, and the SFT-8B editor reaches 79.52% with success rates only and 80.32% with initial-policy summaries. On Seen, initial-policy summaries fall 6.61 points below success rates alone (74.02% versus 80.63%): outdated behavioral evidence is worse than none.

The three feedback settings share validation and candidate ranking, and their gap lies in edit content: initial-policy summaries land revisions in the band at least as often as current-learner summaries (Appendix C.2). The band governs how hard a revision is; the trajectory summary shapes which conditions change, as intended in Section 2.2. Opus editing likewise exceeds the fixed curriculum and random operations. Opus edits pay off through online revision: offline augmentation, a fixed pool of 370 original and Opus-edited versions sampled uniformly, reaches 64.46%, below static training (Appendix C).

4.2 When can world revisions stop?

Figure 3(b) compares three branches that stop editing at learner steps 75, 150, or 225. Each inherits the learner, optimizer, and world pool at its fork and continues training on fixed worlds through step 300. Unseen gains accrue during the first half of training: extending editing from step 75 to step 150 raises final Unseen success from 79.90% to 84.65%, and stopping at step 225 gives the same 84.65%. The stopping point shapes Unseen success; Room success varies within three points without a trend (79.63%, 76.83%, and 79.02%). The branch protocol and paired comparisons appear in Appendix C.

Table 3: Ingredients of online world editing on ALFWorld (success, %; 100 original worlds, 300 updates). Rows differ in whether worlds are edited, what proposes the edits, and whether the success estimate $\widehat{p}$ and trajectory summary $\Sigma(\mathcal{T})$ in Eq. (1) come from the current learner or the initial policy π_{θ_0}. Online configurations share validation, world selection, and candidate ranking with the current learner; the fixed curriculum orders the original worlds by initial-policy success. The last row is the SFT-8B configuration in Table 1. Bold and underlining mark the highest and second-highest scores.

Configuration	Edits	Proposals	$\widehat{p}$	$\Sigma(\mathcal{T})$	Unseen	Seen	Room
Static environments	None	–	–	–	67.21	68.48	69.98
Fixed curriculum	None	–	–	–	77.61	79.33	77.88
One-time editing	Once	SFT-8B	π_{θ_0}	π_{θ_0}	78.31	76.43	76.44
Random operations	Online	Random	Current	–	<u>80.36</u>	78.62	<u>79.16</u>
Success rate only	Online	SFT-8B	Current	–	79.52	<u>80.63</u>	78.86
Initial-policy summaries	Online	SFT-8B	Current	π_{θ_0}	80.32	74.02	77.70
`ResponsiveEnv` w/ SFT-8B	Online	SFT-8B	Current	Current	**85.77**	**80.71**	**81.25**

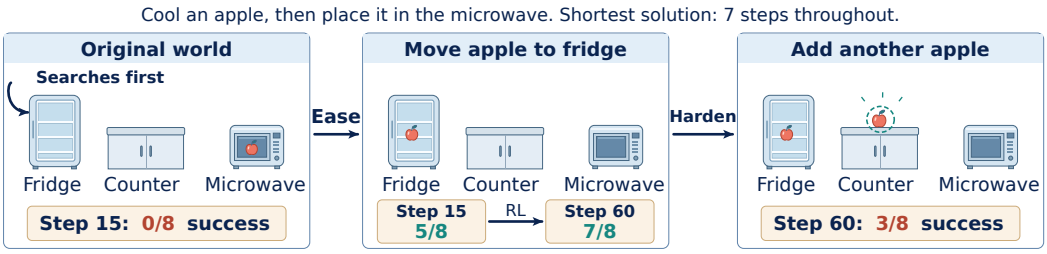

Figure 4: **One task, successive world revisions.** A frozen Opus editor relocates an apple, then adds another after further learner training. Fractions are independent eight-rollout measurements at the indicated learner steps. Probe traces and complete world histories appear in Appendix E.

4.3 What changes when the same world is revisited?

Figure 4 follows one cooling task through two edits by a frozen Opus editor. At learner step 15, moving the apple from the microwave to the fridge changes fresh success from 0/8 to 5/8, with the shortest valid solution fixed at seven steps. All 16 candidate-probe trajectories retrieve the apple; seven execute the cooling action and succeed, while nine omit it and fail. The edit lets the learner reach the stage at which the required skill can be practiced.

By step 60, further training has made this version easy for the learner (7/8 successes). Adding another apple on the countertop changes fresh success to 3/8, again without changing the shortest solution. In the separate candidate probes, eight trajectories cool an apple and succeed; the other eight omit cooling and fail. This is why **`ResponsiveEnv`** revisits accepted worlds as learning proceeds.

A cleaning case provides a complementary observation: moving a kettle onto the learner's search route increases the shortest solution from six to seven steps, yet fresh success rises from 0/8 to 5/8. These cases illustrate behavior-dependent difficulty; Section 4.1 measures its downstream effect. Appendix E reports aggregate editing statistics, case selection, complete world histories, and supporting action sequences.

5 Related Work

Environment scaling and adaptive curricula. Research on training environments addresses both the supply of experience and its adaptation to the learner (Huang et al., 2025; Li et al., 2026). Environment and task synthesis (Hu et al., 2025) and executable tool worlds (Song et al., 2026; Wang et al., 2026b) broaden this supply, and domain randomization and procedural generation vary environment conditions to improve generalization (Tobin et al., 2017; Cobbe et al., 2020). Adaptive curricula match training tasks to the learner (Narvekar et al., 2020; Portelas et al., 2020): they vary starting states for fixed goals (Florensa et al., 2017) or prioritize levels with learning potential (Jiang et al., 2021). POET co-evolves environments with the agents that solve them (Wang et al., 2019), PAIRED uses regret to guide environment generation (Dennis et al., 2020), and ACCEL combines

level editing with regret-based selection (Parker-Holder et al., 2022). For language models, RLVE adapts the difficulty of verifiable environments to the policy (Zeng et al., 2026). For LLM agents, DreamGym couples synthesized transitions and feedback with adaptive tasks (Chen et al., 2026), WebRL generates and filters instructions from failed tasks (Qi et al., 2025), and Agent-World uses failure diagnosis to guide environment and task expansion (Dong et al., 2026). These approaches add worlds, reorder them, or generate new tasks. **`ResponsiveEnv`** instead keeps the original tasks fixed and revises the conditions under which they are practiced. Our coverage and curriculum comparisons (Sections 3.3 and 4.1) separate these effects, and our random-operation control resembles ACCEL-style mutation with learner-based selection.

LLM-guided environment generation and editing. Language models provide semantic knowledge for proposing changes to training environments. EnvGen adapts simulator configurations from agent performance feedback, Eurekaverse generates environment curricula as code, and OMNI-EPIC programs new environments guided by models of interestingness (Zala et al., 2024; Liang et al., 2025; Faldor et al., 2025). Recent methods also learn environment generators: GenEnv co-trains a task generator whose reward peaks at intermediate agent success, while SPADE learns executable environment design through designer–solver self-play and hint-based regret (Guo et al., 2025; Liu et al., 2026). Existing environments can be adapted through critic-guided task perturbations in RLAnything (Wang et al., 2026a) or trajectory-conditioned interface components in EnvHarness (Huang et al., 2026), and Harness-R1 learns to edit an agent's runtime harness from failure trajectories to improve a frozen agent (Shao et al., 2026). Environment Evolution constructs harder terminal environments independently of target-policy rollouts and schedules successive generations during training (Fan et al., 2026). **`ResponsiveEnv`** differs in three respects: it keeps each task's goal fixed and edits only world conditions; revised worlds persist and are revised again as training proceeds; and edits are conditioned on the current learner's trajectories rather than on success rates alone, which Section 4.1 shows matters.

6 Discussion and Limitations

What environment efficiency measures. Environment efficiency compares configurations at a fixed number of learner updates. At this budget, editing trades learner interaction for original worlds: with the 8B editor, a run uses 1.9 times the learner episodes and 1.7 times the wall-clock time of static training, and its 100 worlds outperform a static pool of 2,288 (Appendix A.6).

Scope and uncertainty. Our study uses one learner model, Qwen3-8B, and two text-based benchmarks. We report one training run per configuration, and our intervals capture evaluation uncertainty rather than variation across training runs. On ALFWorld, we edit a 100-world pool. In a much larger pool, most worlds lack recent trajectories, and editing results on the 2,288-world pool varied across editors. Room overlaps the official splits; removing its six worlds that appear in editor training data lowers every configuration's Room success by less than 0.5 points and leaves the ranking unchanged (Table 15). On τ^2-bench, macro gains lie within evaluation noise on fixed test subsets with a local simulator.

Preserving tasks while changing practice. Goal preservation and solvability define what an editor may change; learner probes and held-out endpoints measure whether a change helps. Extending **`ResponsiveEnv`** to a new domain requires specifying its task intent, editable conditions, and validators, as we do for ALFWorld and τ^2-bench.

7 Conclusion

`ResponsiveEnv` revises executable training worlds as the learner changes, while keeping each task's goal fixed. On ALFWorld, editing 100 original worlds outperforms static training on 2,288 worlds at the same number of learner updates. The gains require repeated revision guided by the current learner. Editing once, random edits, success-only feedback, and initial-policy summaries each recover only part of them. The target success band sets how hard a revised world is; the benefit depends on which of its conditions change for the current learner. On τ^2-bench, the same framework carries over to conversational tool use.

USE OF GENERATIVE AI

In this work, we used generative AI tools: Claude and ChatGPT to edit text, tables, and fix grammar issues. The large language models that serve as environment editors are part of the proposed method and are described in Section 2 and Appendix A. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The supplementary material contains anonymous source code for the ALFWorld experiments: the training environment, the world-editing tools and validators, the external, local, and random editors, the training drivers for static environments and **`ResponsiveEnv`**, the learner-feedback ablations, and the endpoint evaluation, with installation instructions that pin every software version and the lists of the 100 training worlds and the 400-instance Room set. Section 2 and Algorithm 1 define the editing loop. Appendix A specifies the training and evaluation settings, editor recipes and deployment, the ALFWorld editing constraints and acceptance rules, world selection, the τ^2-bench editing contracts, the baseline adaptations, resource use, and the uncertainty estimates, and Appendix G gives the editor prompts and tool interfaces. All evaluations use original, unedited tasks with fixed per-attempt seeds, and Appendix B reports endpoint scores for all configurations and the paired comparisons.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 41–48, 2009. URL https://dl.acm.org/doi/10.1145/1553374.1553380.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Xuefei Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason E. Weston, and Dat Huynh. Scaling agent learning via experience synthesis. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=cf7qpBwttr.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056, 2020. URL https://proceedings.mlr.press/v119/cobbe20a.html.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13049–13061, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-World: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026. URL https://arxiv.org/abs/2604.18292.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. OMNI-EPIC: Open-endedness via models of human notions of interestingness with environments programmed in code. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Y1XkzMJpPd.

Zhiyuan Fan, Tinghao Yu, Yuanjun Cai, Jiang Zhou, Jiangtao Guan, Jincheng Liu, Yun Yang, Dingxin Hu, Zhuo Han, Xing Wu, Feng Zhang, and Lilin Wang. Environment evolution for terminal agents. *arXiv preprint arXiv:2609.04128*, 2026. URL https://arxiv.org/abs/2609.04128.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1544. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 482–495, 2017. URL https://proceedings.mlr.press/v78/florensa17a.html.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. GenEnv: Difficulty-aligned co-evolution between LLM agents and environment simulators. *arXiv preprint arXiv:2512.19682*, 2025. URL https://arxiv.org/abs/2512.19682.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. AgentGen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, pp. 496–507, 2025. doi: 10.1145/3690624.3709321. URL https://doi.org/10.1145/3690624.3709321.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi Mishra, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. EnvHarness: Awakening static worlds for agent learning. *arXiv preprint arXiv:2608.19880*, 2026. URL https://arxiv.org/abs/2608.19880.

Yuchen Huang, Sijia Li, Minghao Liu, Wei Liu, Shijue Huang, Zhiyuan Fan, Hou Pong Chan, and Yi R. Fung. Environment scaling for interactive agentic experience collection: A survey. *arXiv preprint arXiv:2511.09586*, 2025. URL https://arxiv.org/abs/2511.09586.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950, 2021. URL https://proceedings.mlr.press/v139/jiang21b.html.

Jiachun Li, Zhuoran Jin, Tianyi Men, Yupu Hao, Kejian Zhu, Lingshuai Wang, Dongqi Huang, Longxiang Wang, et al. Agentic environment engineering for large language models: A survey of environment modeling, synthesis, evaluation, and application. *arXiv preprint arXiv:2606.12191*, 2026. URL https://arxiv.org/abs/2606.12191.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Environment curriculum generation via large language models. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 433–454, 2025. URL https://proceedings.mlr.press/v270/liang25a.html.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, Mickel Liu, Hanfei Yu, Zhaorun Chen, Weiyan Shi, Paul Pu Liang, Luke Zettlemoyer, Yejin Choi, and Natasha Jaques. SPADE: Self-play in adaptive synthetic executable environments. *arXiv preprint arXiv:2608.19197*, 2026. URL https://arxiv.org/abs/2608.19197.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020. URL https://jmlr.org/papers/v21/20-212.html.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17473–17498, 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep RL: A short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 4819–4825, 2020. URL https://www.ijcai.org/proceedings/2020/0671.pdf.

Prime Intellect. PRIME-RL. Software, 2025. URL https://github.com/PrimeIntellect-ai/prime-rl. Accessed September 25, 2026.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, pp. 79791–79821, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c66e1fcc9691aae706250638f36f681b-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2608.02276.

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

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.

David Silver and Richard S. Sutton. Welcome to the era of experience. Preprint, 2025. URL https://storage.googleapis.com/deepmind-media/Era-of-Experience%20/The%20Era%20of%20Experience%20Paper.pdf.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026. doi: 10.18653/v1/2026.findings-acl.407. URL https://doi.org/10.18653/v1/2026.findings-acl.407.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 23–30, 2017. URL https://doi.org/10.1109/IROS.2017.8202133.

Lei Wang, Chen Ma, Xueyang Feng, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18:186345, 2024. URL https://link.springer.com/article/10.1007/s11704-024-40231-1.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019. URL https://arxiv.org/abs/1901.01753.

Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. RLAnything: Forge environment, policy, and reward model in completely dynamic RL system. *arXiv preprint arXiv:2602.02488*, 2026a. URL https://arxiv.org/abs/2602.02488.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent World Model: Infinity synthetic environments for agentic reinforcement learning. In *International Conference on Machine Learning*, 2026b. URL https://arxiv.org/abs/2602.10090.

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

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68:121101, 2025. URL https://link.springer.com/article/10.1007/s11432-024-4222-0.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Jiaqi Liu, Honglin Guo, Yajie Yang, Rui Zheng, Junjie Ye, Jiazheng Zhang, et al. AgentGym-RL: An open-source framework to train LLM agents for long-horizon decision making via multi-turn RL. In *International Conference on Learning Representations*, pp. 12685–12727, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1571ce1ff735be4551db50887043726e-Abstract-Conference.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. EnvGen: Generating and adapting environments via LLMs for training embodied agents. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=F9tqgOPXH5.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. RLVE: Scaling up reinforcement learning for language models with adaptive verifiable environments. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2511.07317.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, et al. The landscape of agentic reinforcement learning for LLMs: A survey. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=RY19y2RI1O.

APPENDIX FOR RESPONSIVEENV

A EXPERIMENTAL SETUP

A.1 TRAINING POOLS AND EVALUATION SETS

The initial ALFWorld pool contains 100 worlds from the official training split, selected without filtering for learner success. It spans 60 rooms and all six task families. The 500- and 2,288-world pools retain these worlds and use the same room boundary. The largest pool excludes rooms used by the additional Room evaluation and the dedicated editor unseen-world set. It is the full eligible pool for this study, rather than the complete official training split of 3,553 worlds. World identity includes both task directory and trial identifier.

Table 4: Original learner-training world counts by ALFWorld task family.

Family	100 worlds	500 worlds	2,288 worlds
Pick	27	135	383
Pick2	22	110	440
Clean	21	105	480
Heat	12	60	393
Cool	11	55	417
Look	7	35	175
Total	100	500	2,288

The 400-world Room set contains 335 instances from the official training split, 20 from Seen, and 45 from Unseen. Its 52 rooms are disjoint from learner training rooms. The official splits and Room therefore overlap each other. All three supervised editor recipes in Table 6 contain data from six of the Room world IDs, all in the Pick family. None of their ALFWorld world IDs overlap official Seen or Unseen instances. The sensitivity analysis removes the same six worlds from every configuration.

A.2 LEARNER AND EVALUATION SETTINGS

ALFWorld evaluation uses top-$p = 0.8$, top-$k = 20$, and the same explicit reasoning-and-action protocol (Yao et al., 2023) as learner training. The chat template's automatic thinking mode is disabled. The τ^2-bench initialization uses 296 self-generated trajectories on 74 telecom training tasks, rendered under the RL conversation convention, followed by 40 SFT updates. All compared τ^2-bench learner runs share this initialization. The user simulator uses `Qwen/Qwen3.8-Flash-Next-FP8`, served locally with SGLang using four-way tensor and expert parallelism and a 32,768-token context. Its thinking mode is disabled. Retail also uses the configured GPT-4.1 judge at temperature zero. These fixed subsets and local simulation settings differ from the full official leaderboard protocol.

A.3 EDITOR RECIPES AND DEPLOYMENT

For the 100-world ALFWorld experiments, archived editor access records are dated September 7 to 9, 2026 for Opus and September 9 to 10, 2026 for GPT and Gemini (UTC). These dates identify the recorded service-access windows.

All local editors use the Qwen3-8B backbone. The default SFT-8B recipe is trained on joint ALFWorld and τ^2-bench examples, with a diagnosis followed by a structured edit list. SFT-8B (augmented) adds editing examples from more learner checkpoints and uses edits-only supervision. Its 1,223 examples contain 795 ALFWorld and 428 τ^2-bench examples, split into 1,091 training and 132 validation examples. This recipe is distinguished by both data and output target, rather than by an internal version number.

The conversational Opus-distillation recipe reuses the default recipe's 505 ALFWorld examples and replaces its 415 Gemini-derived conversational examples with 285 Opus-derived examples. Telecom/retail/airline counts change from 309/86/20 to 175/90/20. Both the teacher and selected data therefore change.

Table 5: Learner settings. The τ^2-bench user simulator introduces variation even when the learner is evaluated at temperature zero.

Setting	ALFWorld	τ^2-bench
Learner backbone	Qwen3-8B	Qwen3-8B
Conversation adaptation before RL	None	Self-distillation
Learner updates	300	300
Trajectories per learner batch	64	64
Trajectories per world group	8	8
Learning rate	1e-6	1e-6
Training context limit	32,768	32,768
Training episode turn limit	30	120
Per-response generation limit	640	2,048
Evaluation episode turn limit	45	120
Evaluation temperature	0.7	0.0
Evaluation attempts per instance	16	16

Table 6: Supervised editor recipes. Counts include the stated training and validation portions. The augmented recipe changes both data composition and output supervision.

Recipe	Output target	ALFWorld	τ^2	Train	Val.
SFT-8B	Diagnosis + edits	505	415	823	97
SFT-8B (augmented)	Edits only	795	428	1,091	132
SFT-8B (Claude Opus 4.8, τ^2)	Diagnosis + edits	505	285	702	88

Supervised editor training. We fine-tune all parameters of Qwen3-8B with prime-rl on eight GPUs for two nominal epochs, using a global batch size of 16, a per-device microbatch size of one, and an 8,192-token sequence limit. AdamW uses a constant learning rate of 1e-5, $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 0.01, and gradient-norm clipping at 1.0. Only assistant tokens contribute to the loss; data are shuffled with seed zero. The default, augmented, and conversational Opus-distillation recipes run for 103, 137, and 88 updates, respectively.

ALFWorld editing constraints. In ALFWorld, edits relocate, add, or remove objects, open or close receptacles, or change the agent's starting location. The PDDL goal and native action semantics remain unchanged. Each edited world is recompiled and checked for solvability by a planner, with additional constraints preventing trivial winning states at reset. These edits can change the grounded state space and solution length while retaining the original goal. The same task can therefore require different searches, object choices, or action sequences across world versions.

Proposals and candidate budgets. External ALFWorld editors use interactive tool sessions. Local editors generate complete candidate lists in one response. Local-editor deployment uses up to four candidates per world. Exact per-run settings are retained in the configuration artifacts. The augmented recipe also uses four candidates. The random-edit and RLAnything-protocol controls also use four candidates. In the single-response pipeline, each candidate is ranked with eight learner attempts and the selected world receives eight fresh post-edit attempts. A fresh pre-edit check uses eight attempts to skip worlds already inside the target band.

Interactive sessions can inspect, modify, probe, and restore candidates; their budgets are not equivalent to a fixed number of independent lists. The prompts and available tools are given in Appendix G. Generation, probing, validation, and local-editor training are additional costs beyond the shared learner-update budget.

World selection and editing schedule. ALFWorld editing takes place between segments of 15 learner updates. Within each segment, training passes through the 100-world pool in shuffled order before any world repeats, so every world is trained in every segment. The editing runs use these recent training trajectories to estimate success and identify worlds for revision. When eligible worlds exceed the editing budget, selection prioritizes distance from $p^\star = 0.5$, reserving part of the budget

for simplification. Local-editor and random-edit runs also use a fresh pre-edit check to skip worlds already inside the target band.

Single-response selection. The local editor samples K independent edit lists from the same current world and learner-feedback packet. Each list is applied to a separate copy of the world; unchanged outputs and invalid revisions are discarded. For $K > 1$, valid candidates are ranked by $q(w;\theta) = -|\widehat{p}_\theta(w, g) - 0.5|$, using fresh learner rollouts. Ties are broken by fewer proposed edits. For $K = 1$, the sole valid revision is selected without a candidate-ranking pass. If no valid revision is available, the current world is retained. The selected revision is measured with separate post-edit rollouts. The learner checkpoint remains fixed throughout the editing evaluation.

ALFWorld acceptance and post-edit measurement. The single-response pipeline selects the highest-scoring valid proposal; the unmodified world is not automatically included among the candidates. Interactive `COMMIT` requires solvability and a minimum probe count. If a session ends without a successful `COMMIT`, the driver can retain its last modified, solvable state without enforcing that probe minimum. Both paths still undergo final goal-preservation and compilation checks. Neither entry into the target band nor a positive Δq is required for persistence. Post-edit success is measured independently and does not reverse acceptance. Appendix E reports the frequency of automatic retention in the main Opus run.

Difficulty target. The target band is $[0.35, 0.65]$ with point target 0.5. For $G > 1$ independent Bernoulli outcomes with success probability p, the probability of both success and failure in a group is $1 - p^G - (1 - p)^G$, maximized at $p = 0.5$. This motivates intermediate difficulty as a practical selection heuristic. The band width is a design choice. This argument establishes neither an optimal band width nor optimal downstream learning progress.

A.4 τ^2-BENCH EDITING CONTRACTS AND EXECUTION

Retail and airline edits modify surrounding database records, distractor alternatives, and removable search clutter through database overlays. Validators protect task-relevant records and replay reference actions to check that their outcomes are preserved. Telecom edits add, remove, or swap faults while preserving the customer's intent, persona, and whether the request is fixable. The benchmark's original generator instantiates the corresponding reference actions and checks for each revised fault composition. The required action sequence may therefore change even though the task intent and evaluation procedure are retained.

At each editing round, success estimates use recent training trajectories of the current world version. The domain-specific function $f_{\text{menu}}(w, g, d)$ constructs a menu of valid revisions for world w and goal g in direction d (simplify or harden). The editor receives the world conditions, task information, success rate, selected interaction excerpts, termination outcomes, and this menu. Both external and local editors choose from it. The chosen valid change is committed directly and then measured with fresh learner rollouts. These post-edit measurements do not gate acceptance. This interface uses neither interactive learner probing nor the single-response candidate-ranking procedure used in ALFWorld.

The editing worker maintains the current pool $\mathcal{W}$ and per-world version IDs v_i alongside continuous updates to the learner policy π_θ. After a revision is published, subsequent episodes load the new world version, while rollout records retain their version identifiers. Later editing rounds exclude trajectories from older world versions when estimating current success. Appendix G provides the domain-specific prompts and output formats. Algorithm 2 summarizes the editing worker that runs alongside learner training.

A.5 BASELINE ADAPTATIONS

EnvHarness. We use the authors' released EnvHarness source code,[1] including its ALFWorld designer and rule-execution components, to construct and apply a fixed mutation corpus for the same 100 initial worlds under our learner protocol. The released RL path likewise loads a precomputed

[1] https://github.com/google-research/envharness

Algorithm 2 **`ResponsiveEnv`**: online editing worker for τ^2-bench

Require: Current pool $\mathcal{W} = \{(w_i, g_i)\}_{i=1}^{N}$ and version IDs $\{v_i\}_{i=1}^{N}$
Require: Fixed editor e_ϕ, band $\mathcal{B}$, target $p^\star$, budget M, minimum trace count m
Require: Access to learner training traces and the current policy π_θ

while learner training continues **do**
  Wait for the next completed training window
  Read recent learner trajectories $\mathcal{T}$
  Filter $\mathcal{T}$ by current version IDs v_i and group traces into $\mathcal{T}_i$
  Estimate success $\widehat{p}_i$ for worlds with at least m retained traces
  $\mathcal{I} \leftarrow f_{\text{select}}(\mathcal{W}, \mathcal{T}, M, \mathcal{B}, p^\star)$; $\mathcal{R} \leftarrow \emptyset$
  for all $i \in \mathcal{I}$ **do**
    $(w, g) \leftarrow \mathcal{W}[i]$
    Set direction d_i to simplify or harden using $\widehat{p}_i$
    $\mathcal{M} \leftarrow f_{\text{menu}}(w, g, d_i)$
    if $\mathcal{M} \neq \emptyset$ **then**
      Construct context x from $(w, g, \mathcal{T}_i, \widehat{p}_i, \mathcal{M})$
      $k \sim e_\phi(\cdot \mid x)$
      if k identifies a valid menu entry **then**
        $w' \leftarrow \mathcal{M}[k]$.world
        $\mathcal{W}[i] \leftarrow (w', g)$ ▷ Persist for training and later revision
        $v_i \leftarrow v_i + 1$; $\mathcal{R} \leftarrow \mathcal{R} \cup \{i\}$
        Publish the revised world and version for subsequent learner episodes
      end if
    end if
  end for
  for all $i \in \mathcal{R}$ **do**
    $(w, g) \leftarrow \mathcal{W}[i]$
    Record $f_{\text{probe}}(\pi_\theta, w, g)$ ▷ Post-publication diagnostic only
  end for
end while

corpus and reapplies its programs at episode reset. State-dependent rule hooks remain active during rollouts; the corpus is not regenerated from later learner checkpoints. The Flash-Lite corpus accepts mutations for 55 worlds: 28 initial-state edits, 21 combined initial-state and rule edits, and six rule edits. The GPT corpus accepts 38 worlds, all initial-state edits. The released instructions permit acceptance based on either a numeric success improvement or evidence that a diagnosed recovery skill appears in the rollouts. Both corpora are evaluated under our 300-update learner training recipe. The comparison changes intervention interface, revision frequency, accepted coverage, and editing expenditure.

RLAnything editing protocol. We port the ALFWorld environment-modification component from the authors' released RLAnything source code[2] into our training pipeline. A frozen Qwen3-8B or GPT proposes changes to goal object and receptacle types in existing worlds. It adopts the task-modification protocol, rather than the complete policy and reward-model training system. We synchronize the natural-language instruction with the rewritten goal, validate candidates with the planner, and use fresh learner attempts for selection before subsequent training. Candidates are accepted when measured success moves in the requested direction without crossing the opposite boundary of the target band. The adaptation uses our batch size, learner-update budget, four-candidate interface, and final evaluation protocol. The GPT configuration enables a comparison with the same proposal model as **`ResponsiveEnv`** / GPT. It retains the single-response goal-editing protocol, whereas **`ResponsiveEnv`** / GPT uses interactive world editing. The comparison therefore still changes the editing interface, acceptance behavior, and proposal and probe budgets.

A.6 Resource use

Table 7 reports the learner episodes, learner tokens, and wall-clock time of each ALFWorld configuration. All configurations use the same 300 learner updates. Editing adds learner episodes outside these updates: pre-edit checks, candidate probes or ranking, and post-edit measurement. With SFT-8B, the full method uses 1.88 times the learner episodes of static training on 100 worlds. Its token count and

[2] https://github.com/Gen-Verse/Open-AgentRL

Table 7: Resource use of each ALFWorld configuration. All configurations take 300 learner updates with 64 episodes each; editing adds learner episodes for pre-edit checks, candidate probes or ranking, and post-edit measurement. Parentheses give ratios to static training on 100 worlds. Hours include editing and are measured on one node with eight NVIDIA H20 GPUs. Editing-side tokens are estimated from episode counts.

Configuration	Editing episodes	Total episodes	Tokens (M)	Hours
Static, 100 worlds	–	19,200 (1.00)	190 (1.00)	11.4 (1.00)
Static, 2,288 worlds	–	19,264 (1.00)	197 (1.04)	12.6 (1.10)
ResponsiveEnv w/ Claude Opus 4.8	25,158	44,422 (2.31)	418 (2.20)	29.0 (2.53)
ResponsiveEnv w/ GPT-5.6 Sol	26,753	45,961 (2.39)	431 (2.27)	23.7 (2.07)
ResponsiveEnv w/ Gemini 3.8 Flash	25,125	44,477 (2.32)	447 (2.35)	23.4 (2.05)
ResponsiveEnv w/ SFT-8B	16,904	36,112 (1.88)	331 (1.74)	19.3 (1.69)
RLAnything w/ Qwen3-8B	8,160	27,568 (1.44)	239 (1.26)	16.9 (1.47)
RLAnything w/ GPT-5.6 Sol	7,360	26,824 (1.40)	256 (1.35)	15.8 (1.38)
Fixed curriculum	–	19,792 (1.03)	159 (0.84)	10.4 (0.91)
One-time editing w/ SFT-8B	3,896	23,096 (1.20)	229 (1.21)	12.7 (1.11)
Random-edit control	19,560	38,832 (2.02)	355 (1.87)	23.3 (2.03)
ResponsiveEnv w/ SFT-8B, success rate only	19,040	38,256 (1.99)	345 (1.82)	18.8 (1.64)
ResponsiveEnv w/ SFT-8B, initial-policy trajectories	18,400	37,672 (1.96)	342 (1.80)	19.1 (1.67)

wall-clock time are 1.74 and 1.69 times those of static training, respectively. Interactive editors probe more, and the RLAnything protocol probes less. The online-editing controls in Section 4.1 use at least as many learner episodes as the full method, 1.96 to 2.02 times those of static training. Their lower success therefore does not come from a smaller probing budget.

A.7 Success estimates and uncertainty

For N instances and $R = 16$ attempts, success is

$$\widehat{S} = \frac{100}{N} \sum_{i=1}^{N} \left(\frac{1}{R} \sum_{r=1}^{R} V(\tau_{i,r}) \right). \tag{4}$$

Appendix B lists endpoint scores for all configurations.

We use 20,000 instance-bootstrap resamples. For comparisons, the same sampled instance IDs are applied to both configurations. The ALFWorld snapshot and figures use bootstrap seed 20260922; the inherited τ^2-bench and local summary-probe analyses use 20260911. Aggregate τ^2-bench intervals resample within each domain and apply the stated domain or task weights. Missing task-success outcomes in its audited logs remain in the fixed denominator as failures.

Intervals condition on the trained checkpoints and their observed attempts. They do not capture training-run variability, and no multiple comparison adjustment is applied. A difference interval containing zero is not an equivalence test.

B Additional Endpoint Results

B.1 Endpoint scores for all configurations

Table 8 compares static pools of 100 to 2,288 original worlds with editing of the 100-world pool.

Tables 9 and 10 list all ALFWorld configurations.

Table 8: Original environment coverage. Scores are success (%) after 300 learner updates. Static rows train on 100 to 2,288 unedited worlds; editing rows revise the 100-world pool. Bold and underlining mark the highest and second-highest point estimates in each score column.

Configuration	Worlds	Unseen	Seen	Room
Static environments	100	67.21	68.48	69.98
Static environments	500	82.32	76.16	78.02
Static environments	2,288	82.74	76.79	76.11
ResponsiveEnv w/ Claude Opus 4.8	100	**86.15**	**82.32**	<u>82.09</u>
ResponsiveEnv w/ GPT-5.6 Sol	100	85.35	80.40	**83.13**
ResponsiveEnv w/ SFT-8B	100	<u>85.77</u>	<u>80.71</u>	81.25

Table 9: ALFWorld configurations trained on the original 100-world task pool (success, %). Bold and underlining mark the highest and second-highest point estimates in each column.

Configuration	Unseen	Seen	Room
Static, 100 worlds	67.21	68.48	69.98
EnvHarness w/ Gemini 3.1 Flash-Lite	78.54	80.49	78.52
EnvHarness w/ GPT-5.6 Sol	69.22	67.77	71.31
RLAnything w/ Qwen3-8B	**86.57**	76.65	80.22
RLAnything w/ GPT-5.6 Sol	82.00	78.48	78.41
ResponsiveEnv w/ Gemini 3.8 Flash	75.42	72.41	75.27
ResponsiveEnv w/ GPT-5.6 Sol	85.35	80.40	**83.13**
ResponsiveEnv w/ Claude Opus 4.8	<u>86.15</u>	**82.32**	<u>82.09</u>
ResponsiveEnv w/ SFT-8B	85.77	<u>80.71</u>	81.25
Fixed curriculum	77.61	79.33	77.88
Offline augmentation w/ Claude Opus 4.8	64.46	65.58	67.50
Random-edit control	80.36	78.62	79.16
One-time editing w/ SFT-8B	78.31	76.43	76.44
ResponsiveEnv w/ SFT-8B, success rate only	79.52	80.63	78.86
ResponsiveEnv w/ SFT-8B, initial-policy trajectories	80.32	74.02	77.70

B.2 Paired main and coverage comparisons

Table 11 reports paired differences for the main baseline and environment-coverage comparisons across all three evaluation splits. Strategy, feedback, and stopping comparisons appear in Tables 19, 20, and 23. All intervals are conditional on the reported checkpoints.

B.3 Task-family performance

Tables 12, 13, and 14 report success for all six ALFWorld task families on Unseen, Seen, and Room, respectively. The n row gives the number of instances per family. The final column weights instances equally, with the same training-run averaging as the aggregate results.

B.4 Excluding known editor-data overlap

The two ALFWorld SFT recipes and the Opus-derived conversational recipe each contain examples from the same 145 original ALFWorld world IDs. Their data overlap neither official Seen nor official Unseen evaluation instances. Six IDs overlap the additional Room set. Table 15 removes these six instances for every configuration, retaining the same common set of 394 worlds. It reports both the original and reduced-set scores so that the sensitivity can be assessed for each configuration.

B.5 Additional τ^2-bench aggregates

Opus improves the task-weighted mean over the static reference by 4.25 points with a 95% interval $[-0.25, 8.69]$. Its domain-macro gain is 3.85 points $[-1.30, 8.80]$. For SFT-8B with Opus-derived conversational supervision, the corresponding gains are 3.69 points $[-0.63, 7.94]$ and 3.75 points

Table 10: Additional ALFWorld coverage configurations (success, %). Bold and underlining mark the highest and second-highest point estimates in each column.

Configuration	Unseen	Seen	Room
Static, 500 worlds	<u>82.32</u>	<u>76.16</u>	**78.02**
Static, 2,288 worlds	**82.74**	**76.79**	<u>76.11</u>

Table 11: Paired ALFWorld comparisons in points (95% instance-bootstrap intervals). Parentheses give original training-world counts. The EnvHarness and GPT-5.6 Sol RLAnything contrasts use GPT-5.6 Sol in both pipelines. Each interval conditions on the reported trained checkpoints.

Configuration − reference	Unseen	Seen	Room
ResponsiveEnv w/ Claude Opus 4.8 (100) − Static environments (100)	18.94 [14.04, 23.93]	13.84 [9.64, 18.26]	12.11 [9.73, 14.56]
ResponsiveEnv w/ Claude Opus 4.8 (100) − Static environments (500)	3.82 [1.21, 6.34]	6.16 [2.68, 9.82]	4.08 [2.09, 6.06]
ResponsiveEnv w/ Claude Opus 4.8 (100) − Static environments (2,288)	3.40 [0.37, 6.44]	5.54 [2.23, 8.97]	5.98 [3.91, 8.05]
ResponsiveEnv w/ Claude Opus 4.8 − RLAnything w/ Qwen3-8B	−0.42 [−3.78, 2.89]	5.67 [2.10, 9.42]	1.88 [−0.30, 4.06]
ResponsiveEnv w/ GPT-5.6 Sol − EnvHarness w/ GPT-5.6 Sol	16.14 [11.89, 20.38]	12.63 [8.88, 16.70]	11.81 [9.81, 13.91]
ResponsiveEnv w/ GPT-5.6 Sol − RLAnything w/ GPT-5.6 Sol	3.36 [−0.09, 7.00]	1.92 [−1.34, 5.18]	4.72 [2.77, 6.72]

$[-1.09, 8.44]$. At the domain level, Opus improves telecom success by 7.03 points with a 95% interval $[0.16, 13.91]$. The Opus-distilled SFT-8B editor improves retail success by 9.22 points $[2.66, 15.78]$. The Gemini-distilled SFT configuration is an additional editor recipe. Its teacher and data composition differ from the Opus-distilled configuration.

C Training Strategies, Learner Feedback, and Revision Timing

C.1 Fixed curriculum and editing controls

Fixed curriculum. The curriculum uses the same 100 unedited worlds as the static reference. Initial-learner success rates determine a fixed order with seeded tie-breaking. The first 15-update segment uses the easiest 50 worlds. Subsequent segments unlock approximately 5.56% of the pool at a time, so all 100 worlds are available after update 135. Within each segment, the available worlds are allocated as evenly as possible across 100 training slots. The order and unlocking schedule do not use later learner feedback. Batch size, group size, learning rate, and the 300-update budget follow the main configuration.

Offline augmentation. Claude Opus 4.8 receives world states and initial-learner feedback and proposes four independent edit lists per original world, each with at most 12 primitive edits. We retain up to three distinct, solvable variants per original world, excluding unchanged proposals and duplicates. One of the 100 source worlds did not produce a valid initial-learner packet after a planner timeout, leaving 99 originals. Of 396 proposals, 271 are retained, 81 duplicate an existing version or leave it unchanged, 41 exceed the per-world retention limit, and three produce no legal variant. The 99 originals and 271 variants form a fixed pool of 370 world versions, sampled uniformly throughout 300 learner updates. There is no success-rate ranking of the generated candidates and no editing during learner training. This differs from online Opus in pool size, single-response rather than interactive proposals, and candidate selection. Table 18 reports all three evaluation splits; Table 19 gives the corresponding checkpoint-conditional differences.

Random-edit control. Each proposal samples one to four primitive operations from the legal edit interface. The pipeline retains retrospective learner-based world selection, validation, and ranking of four candidates using fresh learner attempts.

Table 12: ALFWorld official unseen success (%) by task family. The final column weights instances equally.

Configuration	Pick	Pick2	Clean	Heat	Cool	Look	All
n	24	17	31	23	21	18	134
Static, 100 worlds	98.44	76.84	64.52	41.85	57.14	65.28	67.21
EnvHarness w/ Gemini 3.1 Flash-Lite	99.74	86.76	86.29	54.62	61.90	79.17	78.54
EnvHarness w/ GPT-5.6 Sol	97.66	68.75	64.31	62.50	53.87	66.67	69.22
RLAnything w/ Qwen3-8B	97.66	73.53	95.77	74.18	84.23	86.81	86.57
RLAnything w/ GPT-5.6 Sol	98.96	72.06	85.48	63.59	88.69	78.47	82.00
ResponsiveEnv w/ Gemini 3.8 Flash	98.18	75.37	79.44	50.82	71.13	74.65	75.42
ResponsiveEnv w/ GPT-5.6 Sol	99.22	88.97	81.25	71.74	87.20	85.76	85.35
ResponsiveEnv w/ Claude Opus 4.8	97.40	81.62	83.47	82.61	88.69	81.60	86.15
ResponsiveEnv w/ SFT-8B	97.40	85.29	82.66	78.26	91.96	78.47	85.77
Fixed curriculum	98.96	72.79	82.46	55.43	75.60	76.04	77.61
Offline augmentation w/ Claude Opus 4.8	98.70	69.85	64.31	30.43	52.98	70.83	64.46
Random-edit control	96.88	78.68	73.99	67.12	94.05	71.88	80.36
One-time editing w/ SFT-8B	99.22	78.31	80.44	52.72	85.42	71.18	78.31
Static, 500 worlds	95.57	80.88	77.42	78.80	88.99	71.18	82.32
Static, 2,288 worlds	98.96	82.35	81.65	77.99	77.38	75.69	82.74
ResponsiveEnv w/ SFT-8B, success rate only	97.66	78.31	71.98	83.15	75.60	69.44	79.52
ResponsiveEnv w/ SFT-8B, initial-policy trajectories	93.23	79.04	76.81	70.92	92.26	68.40	80.32

Table 13: ALFWorld official seen success (%) by task family. The final column weights instances equally.

Configuration	Pick	Pick2	Clean	Heat	Cool	Look	All
n	35	24	27	16	25	13	140
Static, 100 worlds	96.96	90.63	59.03	34.77	44.25	58.65	68.48
EnvHarness w/ Gemini 3.1 Flash-Lite	100.00	97.14	79.63	41.80	62.25	81.73	80.49
EnvHarness w/ GPT-5.6 Sol	99.82	89.84	52.31	43.75	35.00	65.38	67.77
RLAnything w/ Qwen3-8B	99.46	90.89	71.76	36.72	62.75	75.00	76.65
RLAnything w/ GPT-5.6 Sol	99.11	87.50	71.76	52.73	70.50	67.31	78.48
ResponsiveEnv w/ Gemini 3.8 Flash	99.82	92.45	61.11	35.55	55.50	62.98	72.41
ResponsiveEnv w/ GPT-5.6 Sol	99.82	93.49	70.60	46.48	67.75	90.38	80.40
ResponsiveEnv w/ Claude Opus 4.8	98.21	93.49	75.93	64.84	69.50	78.37	82.32
ResponsiveEnv w/ SFT-8B	98.75	91.67	71.30	59.38	74.75	69.23	80.71
Fixed curriculum	100.00	94.27	81.48	34.77	64.25	75.48	79.33
Offline augmentation w/ Claude Opus 4.8	98.93	90.10	50.46	16.41	43.75	64.42	65.58
Random-edit control	100.00	90.89	63.19	58.20	70.75	70.67	78.62
One-time editing w/ SFT-8B	98.93	90.89	71.53	34.38	68.75	65.87	76.43
Static, 500 worlds	98.93	92.45	64.35	49.61	65.00	63.46	76.16
Static, 2,288 worlds	99.46	95.05	63.43	52.34	60.25	71.63	76.79
ResponsiveEnv w/ SFT-8B, success rate only	99.82	94.53	77.55	58.98	66.50	63.46	80.63
ResponsiveEnv w/ SFT-8B, initial-policy trajectories	99.46	88.02	62.04	53.13	55.50	65.87	74.02

One-time editing. SFT-8B edits the original pool once against the initial learner, with up to 100 worlds considered and four candidates per world. The resulting pool remains fixed for 300 learner updates. We compare this endpoint with the SFT-8B configuration in Table 1. The comparison changes the editing schedule, cumulative editing budget, and exposure to revised worlds.

C.2 Learner-feedback controls

Table 3 includes three feedback settings for the same frozen SFT-8B editor, starting from the original 100-world pool and using 300 learner updates. The settings share the edit interface, validation, candidate-ranking rule, and 15-update editing interval. The current-trajectory reference is the SFT-8B endpoint in Table 1; each row reports one training run. The two new feedback settings use four candidates throughout. Exact per-run candidate settings are retained in the configuration artifacts.

Success rate only. The summary retains the number of rollouts and successes and task-object identifiers, but suppresses search order, action-use statistics, and step counts. The world snapshot, task constraints, and current-learner success rate remain available to the editor. Pre-edit gating and candidate ranking continue to use current-learner rollouts.

Table 14: ALFWorld room-held-out success (%) by task family. The final column weights instances equally.

Configuration	Pick	Pick2	Clean	Heat	Cool	Look	All
n	94	94	61	44	59	48	400
Static, 100 worlds	97.07	83.44	59.73	37.22	43.96	65.63	69.98
EnvHarness w/ Gemini 3.1 Flash-Lite	99.40	91.42	73.67	44.32	55.61	77.99	78.52
EnvHarness w/ GPT-5.6 Sol	96.94	82.65	61.89	44.74	44.17	68.62	71.31
RLAnything w/ Qwen3-8B	97.87	88.16	77.77	55.54	61.97	78.26	80.22
RLAnything w/ GPT-5.6 Sol	99.40	84.04	73.67	56.11	62.50	72.27	78.41
ResponsiveEnv w/ Gemini 3.8 Flash	99.60	89.30	67.21	41.48	51.38	70.70	75.27
ResponsiveEnv w/ GPT-5.6 Sol	99.53	92.15	71.21	63.64	66.53	86.72	83.13
ResponsiveEnv w/ Claude Opus 4.8	98.47	86.10	71.93	73.15	68.54	79.95	82.09
ResponsiveEnv w/ SFT-8B	98.74	87.57	73.36	68.75	71.29	68.36	81.25
Fixed curriculum	95.74	85.84	78.07	43.61	61.02	79.17	77.88
Offline augmentation w/ Claude Opus 4.8	98.60	84.38	53.28	17.90	40.04	70.83	67.50
Random-edit control	99.07	89.76	61.17	62.93	66.95	72.14	79.16
One-time editing w/ SFT-8B	98.60	87.23	73.46	37.07	59.75	72.27	76.44
Static, 500 worlds	97.27	87.17	63.11	63.49	62.92	73.18	78.02
Static, 2,288 worlds	99.14	85.31	64.45	55.40	56.67	70.70	76.11
ResponsiveEnv w/ SFT-8B, success rate only	99.34	89.10	67.32	68.04	56.04	71.35	78.86
ResponsiveEnv w/ SFT-8B, initial-policy trajectories	98.47	89.36	67.52	63.07	55.51	67.84	77.70

Table 15: Room-held-out success before and after removing six world IDs overlapping editor SFT data. Each column uses a common instance set across all configurations.

Configuration	Room-400	Room-394
Static, 100 worlds	69.98	69.54
EnvHarness w/ Gemini 3.1 Flash-Lite	78.52	78.19
EnvHarness w/ GPT-5.6 Sol	71.31	70.88
RLAnything w/ Qwen3-8B	80.22	79.93
RLAnything w/ GPT-5.6 Sol	78.41	78.08
ResponsiveEnv w/ Gemini 3.8 Flash	75.27	74.89
ResponsiveEnv w/ GPT-5.6 Sol	83.13	82.87
ResponsiveEnv w/ Claude Opus 4.8	82.09	81.82
ResponsiveEnv w/ SFT-8B	81.25	80.96
Fixed curriculum	77.88	77.54
Offline augmentation w/ Claude Opus 4.8	67.50	67.01
Random-edit control	79.16	78.84
One-time editing w/ SFT-8B	76.44	76.08
Static, 500 worlds	78.02	77.68
Static, 2,288 worlds	76.11	75.75
ResponsiveEnv w/ SFT-8B, success rate only	78.86	78.54
ResponsiveEnv w/ SFT-8B, initial-policy trajectories	77.70	77.36

Initial-policy trajectories. At each editing opportunity that passes the pre-edit gate, the frozen initial policy π_{θ_0} supplies eight fresh rollouts on the current world version. These replace current-learner rollouts as the source of the behavioral summary. They are not cached trajectories from the original world. The summary format and surrounding instructions are unchanged; its behavioral and outcome statistics describe π_{θ_0}, while the separately supplied success rate describes the current learner. World selection, pre-edit gating, candidate ranking, and post-edit measurement all continue to use the current learner. Thus, this control changes the policy that supplies behavioral evidence while retaining current difficulty estimates in the editing pipeline.

Table 20 reports the three pairwise comparisons. Intervals resample evaluation instances and condition on the reported checkpoints, rather than estimating training-run variability.

Post-edit difficulty. Table 21 counts the worlds edited over the 19 editing rounds of each setting and how often their post-edit success, measured with separate current-learner rollouts, lies in the target band. Initial-policy summaries land in the band at least as often as current-learner summaries, so their lower endpoints do not reflect poorer difficulty calibration.

Table 16: Additional τ^2-bench endpoint aggregates (%). Macro is the unweighted domain mean; Weighted gives each of the 100 tasks equal weight.

Configuration	Telecom	Retail	Airline	Macro	Weighted
Static environments	50.47	49.06	20.94	40.16	44.00
ResponsiveEnv w/ Gemini 3.8 Flash	52.03	53.44	21.56	42.34	46.50
ResponsiveEnv w/ Claude Opus 4.8	57.50	51.72	22.81	44.01	48.25
ResponsiveEnv w/ SFT-8B (Claude Opus 4.8)	48.44	58.28	25.00	43.91	47.69
ResponsiveEnv w/ SFT-8B (Gemini 3.8 Flash)	46.25	50.78	20.63	39.22	42.94

Table 17: Controls in Table 3 and Figure 3(a). Each is compared with the full method, **ResponsiveEnv** with SFT-8B and current-learner feedback. Appendices C and C.2 give the protocols.

Control	Construction
Fixed curriculum	Orders the 100 original worlds by initial-policy success and unlocks them from easy to hard; no world is edited.
One-time editing	SFT-8B edits the original worlds once before training, against the initial policy; the edited pool stays fixed.
Random operations	Random legal edit lists replace the editor's proposals; world selection, validation, and candidate ranking follow the full method.
Success rate only	The SFT-8B editor receives current-learner success rates without behavioral summaries.
Initial-policy summaries	Behavioral summaries come from initial-policy rollouts on the current world; success rates come from the current learner.

C.3 Stopping world revisions

The three branches inherit resumable snapshots at steps 75, 150, and 225 from an Opus parent run. Each snapshot contains learner weights, optimizer state, training progress, and the world versions used in the preceding segment. Snapshots are taken before the editing operation scheduled at that step. Each branch then disables further editing and continues learner updates on its fixed world pool through step 300, using the parent's segment-seed schedule.

The scheduled operation at step 120 was skipped after a service failure. Consequently, the snapshots inherit four, eight, and thirteen active editing rounds, respectively. We identify variants by their stopping update rather than by an assumed number of editing rounds. The paired intervals compare the same instances within each split and condition on these branches; they do not estimate variation across independently trained parent runs.

D Offline Refinement of a Local Editor

This supplementary experiment examines whether editor optimization changes the utility of the worlds supplied to a learner. The initialization is SFT-8B (augmented), whose data and output targets differ from the default SFT-8B recipe (Table 6). We compare refinement with its own supervised initialization.

D.1 Protocol

For a fixed distribution $\mathcal{D}$ of world–learner packets, the editor is optimized with

$$J_{\text{edit}}(\phi) = \mathbb{E}_{x\sim\mathcal{D},\ a\sim e_\phi(\cdot|x)}[r_{\text{edit}}(x,a)]. \tag{5}$$

The reward measures improvement in distance to the target success rate (Equation 2) and penalizes invalid edits. We use the editor after 120 RL updates against fixed learner checkpoints. During each subsequent 300-update learner run, editor weights remain frozen; only the input packets and training worlds change.

Table 18: Training strategies derived from the original ALFWorld pool. All scores are success (%) at learner update 300. Pool size counts world versions in the full training pool; online editing replaces versions within that pool. It does not count cumulative versions seen over training.

Configuration	Pool size	Unseen	Seen	Room
Static, 100 worlds	100	67.21	68.48	69.98
Fixed curriculum	100	77.61	79.33	77.88
Offline augmentation w/ Claude Opus 4.8	370	64.46	65.58	67.50
Random-edit control	100	80.36	78.62	79.16
One-time editing w/ SFT-8B	100	78.31	76.43	76.44
ResponsiveEnv w/ SFT-8B	100	85.77	80.71	81.25
ResponsiveEnv w/ Claude Opus 4.8	100	86.15	82.32	82.09

Table 19: Paired strategy comparisons in points (95% intervals). SFT-8B uses the configuration in Table 1. Intervals resample instances and condition on the listed trained checkpoints.

Configuration − reference	Unseen	Seen	Room
Fixed curriculum − Static environments	10.40 [6.02, 14.65]	10.85 [6.65, 15.09]	7.89 [5.66, 10.16]
ResponsiveEnv w/ Claude Opus 4.8 − Fixed curriculum	8.54 [4.66, 12.55]	2.99 [−0.98, 7.05]	4.22 [1.83, 6.66]
ResponsiveEnv w/ Claude Opus 4.8 − Offline augmentation w/ Claude Opus 4.8	21.69 [16.18, 27.33]	16.74 [12.01, 21.74]	14.59 [11.86, 17.41]
ResponsiveEnv w/ Claude Opus 4.8 − Random-edit control	5.78 [2.24, 9.42]	3.71 [0.63, 6.74]	2.94 [0.81, 5.11]
ResponsiveEnv w/ SFT-8B − Fixed curriculum	8.16 [3.96, 12.50]	1.38 [−2.46, 5.40]	3.38 [0.84, 5.91]
ResponsiveEnv w/ SFT-8B − One-time editing w/ SFT-8B	7.46 [4.06, 10.96]	4.29 [1.03, 7.81]	4.81 [2.63, 6.98]
ResponsiveEnv w/ SFT-8B − Random-edit control	5.41 [2.33, 8.68]	2.10 [−0.67, 5.09]	2.09 [0.28, 3.91]

D.2 Downstream results

At $K = 4$, refinement increases Seen success from 78.13% to 82.86%, a difference of 4.73 points $[1.47, 7.99]$. The Room difference is 3.72 points $[1.72, 5.73]$. The Unseen difference is 1.82 points $[-1.91, 5.64]$.

For the refined editor, using four candidates instead of one raises Seen and Room success by 5.85 points and 5.00 points, respectively. This also increases candidate-generation and evaluation expenditure. The experiment does not include every combination of initialization and candidate count, and each row contains one learner training run. The results describe these particular refinement and search configurations.

E Editing Behavior and Trajectory Evidence

E.1 Aggregate editing behavior

Figure 5 summarizes the frozen-Opus training run. Of 450 editing records, 418 have a completed post-edit measurement. Among these, 279 preserve the shortest solution length, and 151 move the measured success rate closer to 0.5, including 90 whose solution length is unchanged. In total, 140 post-edit measurements fall in $[0.35, 0.65]$, including 93 with unchanged solution length. Thus, 33.5% of completed post-edit measurements lie in the target band, and 66.7% preserve the shortest solution length. These are descriptive statistics from finite-rollout measurements, not estimates of subsequent learning progress.

Of the 418 measured sessions, 392 use an explicit `COMMIT` and 26 (6.2%) use automatic retention. Each automatic case follows a `COMMIT` rejection for insufficient probes, under the fallback described in Appendix A.3. Among explicit commits, 178 have an in-band candidate-ranking estimate; 134 have an in-band independent post-edit estimate. These counts separate the selection measurements

Table 20: Paired feedback comparisons for **`ResponsiveEnv`** w/ SFT-8B in points (95% instance-bootstrap intervals). All rows use the same endpoints as Table 3. Intervals condition on the reported checkpoints.

Feedback contrast	Unseen	Seen	Room
Current trajectories − success rate only	$6.25\ [2.75, 9.98]$	$0.09\ [-3.26, 3.39]$	$2.39\ [0.25, 4.52]$
Current − initial-policy trajectories (π_{θ_0})	$5.46\ [2.43, 8.72]$	$6.70\ [3.93, 9.73]$	$3.55\ [1.66, 5.48]$
Success rate only − initial-policy trajectories (π_{θ_0})	$-0.79\ [-4.90, 3.22]$	$6.61\ [2.50, 10.89]$	$1.16\ [-1.19, 3.52]$

Table 21: Post-edit difficulty under the three feedback settings. Each edited world's post-edit success comes from separate current-learner rollouts; percentages count edited worlds whose post-edit success lies in the band $[0.35, 0.65]$, or moved closer to 0.5.

Feedback supplied to the editor	Edited	Hardened in band	Simplified in band	Closer to 0.5
Current-learner summaries	374	18.1% of 248	24.6% of 126	38.2%
Success rate only	381	21.8% of 266	22.6% of 115	42.5%
Initial-policy summaries (π_{θ_0})	365	23.0% of 261	37.5% of 104	43.3%

from the subsequent difficulty measurements. The counts and source-record indices are retained in `data/editor_acceptance_20260922.json`.

Hardening accounts for 297 of the 450 requests (66.0%). Its share rises from 48.8% over the first 75 updates to 80.0% over the last 75. The completed measurements cover 98 worlds, with a median of four per world. Of the 418 measured sessions, 403 retain at least one successful write operation after checkpoint/restore bookkeeping. Object relocation, agent relocation, and object addition are the three most frequent retained operations. Shortest solution length increases in 77 measurements and decreases in 62. A change in solution length alone does not establish a return to a previous world state.

E.2 Selection and measurement provenance

The cases in Section 4.3 are selected retrospectively from this run. Among 98 worlds with a recorded instance path, nine meet our case-selection criterion: an early pre-edit success of at most 0.25 and a later pre-edit success of at least 0.875, with both corresponding post-edit measurements inside the target band. The apple world is selected from this group. The kettle example is selected to demonstrate a longer planner solution alongside higher learner success. Neither is a random representative sample.

The fractions in Table 25 distinguish two sources: independent eight-rollout pre/post measurements and separate candidate probes used within editing. Figure 4 visualizes the former and summarizes action patterns in the latter. The editor's parameters remain frozen; these cases concern evolving worlds, not online editor parameter learning. They are training-world records, not held-out evaluation trajectories. The aggregate endpoint files do not retain complete actions.

E.3 Cooling: search access, action omission, and revisiting

In the original apple world, the target is in the microwave. The step-15 learner instead begins by searching the fridge and other receptacles, and all eight baseline measurements fail. An initial countertop relocation also fails in all four of its candidate probes. Moving the apple into the fridge then allows all 16 probes to retrieve it. Seven succeed after cooling; nine omit cooling and fail. A recorded successful probe (seed 3005) executes the following complete action sequence:

```
go to fridge 1
open fridge 1
take apple 1 from fridge 1
cool apple 1 with fridge 1
go to microwave 1
open microwave 1
move apple 1 to microwave 1
```

Table 22: Stopping world revisions at three training stages. Each branch inherits a snapshot from the same **`ResponsiveEnv`** w/ Claude Opus 4.8 run and continues on fixed worlds through learner update 300. Scores are success (%).

Stop editing at	Unseen	Seen	Room
Step 75	79.90	79.11	79.63
Step 150	84.65	77.19	76.83
Step 225	84.65	79.69	79.02

Table 23: Differences among the three **`ResponsiveEnv`** w/ Claude Opus 4.8 stopping branches in points (95% paired instance-bootstrap intervals). All branches are evaluated at learner update 300.

Contrast	Unseen	Seen	Room
Stop 150 − stop 75	$4.76\ [1.21, 8.16]$	$-1.92\ [-5.80, 1.88]$	$-2.80\ [-4.84, -0.78]$
Stop 225 − stop 150	$0.00\ [-3.13, 3.22]$	$2.50\ [-0.18, 5.22]$	$2.19\ [0.39, 4.00]$
Stop 225 − stop 75	$4.76\ [1.03, 8.49]$	$0.58\ [-2.72, 3.88]$	$-0.61\ [-2.63, 1.39]$

The representative failed probe (seed 3004) retrieves and places the apple without cooling, then repeats ineffective actions until the 30-step limit. The separate post-commit measurement succeeds in five of eight attempts.

At step 60, the current fridge version achieves seven of eight successes before another edit. No intervening edit of this world appears in the record, although the learner has trained on other worlds. The accepted revision retains the fridge apple and adds another apple on the countertop. All 16 candidate probes retrieve an apple; the eight successes cool it, and the eight failures omit cooling. The additional apple is a legal target candidate, not an irrelevant object. The traces do not establish that failure comes from selecting a particular "wrong apple." Object display IDs can change during world recompilation, so the figure identifies apples by location rather than assuming a stable display number.

E.4 Cleaning: learner difficulty versus planner length

The kettle initially sits on a dining table, while the learner searches cabinets. Moving it to cabinet 5 places it along that search route. All 16 candidate probes retrieve the kettle; eight clean it and succeed, and eight omit cleaning and fail. Fresh success changes from 0/8 to 5/8 even though the shortest solution grows from six to seven actions. The representative successful trajectory first searches cabinets, then executes `take kettle 1 from cabinet 5`, `clean kettle 1 with sinkbasin 1`, and `move kettle 1 to diningtable 1`, with the required navigation in between. Its full 15-action sequence is retained in the data supplement. This case shows that planner length and a particular learner's behavior measure different aspects of difficulty.

E.5 Complete selected-world histories

Table 26 includes every logged edit to the two selected worlds. Later edits do not always meet the local objective: the apple post-edit measurement at step 135 is 2/8, outside the target band. At step 255 the driver automatically commits a version after the explicit commit's probe-count requirement has not been met. These records are kept in the history rather than filtered to show only successful band alignment.

The source supplement `data/trajectory_cases_20260917.json` records world IDs, representative complete action sequences, probe summaries, source hashes, and source-record line numbers. The apple edits are records 15 and 80 of `formal6_responsive_opus_b64.jsonl`; the first kettle edit is record 9. Full histories use the same world identifiers throughout. The diagram is an illustration of logged states, not a simulator screenshot.

Table 24: Offline editor refinement on ALFWorld. All configurations use 100 original worlds and 300 learner updates. Refinement starts from SFT-8B (augmented) and performs 120 editor RL updates against fixed learner checkpoints. Editor parameters are then frozen during each downstream learner run.

Environment editor	K	Unseen	Seen	Room
SFT-8B (augmented)	4	78.59	78.13	78.27
RL-refined SFT-8B	1	79.99	77.01	76.98
RL-refined SFT-8B	4	80.41	82.86	81.98

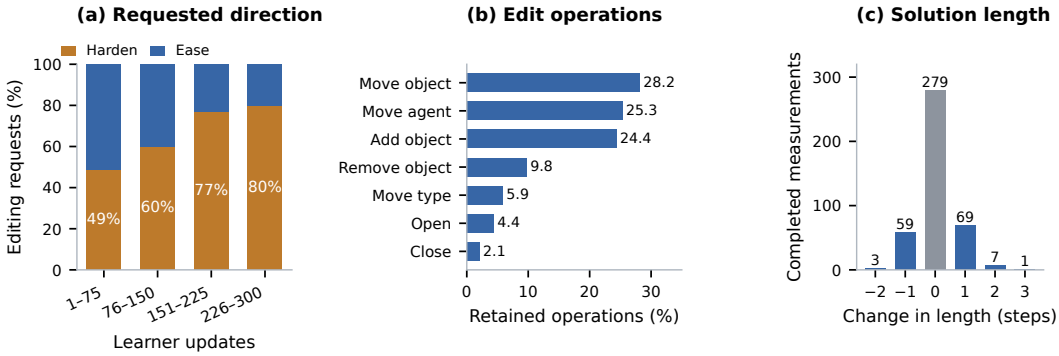

Figure 5: **Aggregate editing behavior in the 100-world Opus run.** (a) Requested directions across 450 editing sessions, grouped by learner stage (125, 100, 125, and 100 sessions). (b) The 1,157 successful write operations retained after checkpoint/restore bookkeeping in measured sessions. (c) Changes in shortest valid solution length for the 418 completed post-edit measurements. Counts describe this run; operation counts do not measure distinct net state changes.

F A Local Probe of Additional Failure Summaries

For each of Opus and GPT, we compare 42 packet evaluations with and without an additional failure-summary field. The primary comparison uses the 25 mixed-outcome packets for which that field changes the input; the other 17 have identical prompts and form a stochastic control. The standard trajectory summary and success estimate remain available in both conditions.

The proportion of edits entering the target band is 32% versus 32% for Opus, and 32% versus 28% for GPT when all 25 attempts are retained. One GPT-on attempt has all proposed edits rejected; it counts as no change and outside the band. Restricting GPT to its 24 complete pairs gives 29.17% in both conditions, with a difference interval $[-20.83, 25.00]$ points. Table 27 uses 20,000 paired packet-bootstrap resamples.

This probe concerns the extra summary field under the tested local protocol. It neither removes all learner feedback nor measures the downstream effect of training on the resulting worlds. Section 4.1 separately evaluates downstream learner training when behavioral summaries are suppressed or supplied by the initial policy.

G Prompts and Interaction Protocols

This appendix documents the learner prompts and the environment-editor interfaces used in the main experiments. We distinguish an *interactive environment editor*, which inspects and modifies an ALFWorld state through tool calls, from a *distilled environment editor*, which produces a structured edit list in one response. In τ^2-bench, both external and distilled editors select from a domain-specific menu of legal changes. The frozen distilled editor receives updated learner feedback at each editing round even though its parameters remain fixed.

Table 28 summarizes the information available to each role. The prompt blocks reproduce the implemented instructions, with runtime fields represented by braced placeholders. Typographic

Table 25: Selected training-world edits. Before/after success uses independent eight-rollout measurements; probes are separate candidate-selection rollouts. L is the shortest valid solution length.

Case	Learner step	Before	After	L before/after	Probe success
Apple, first edit	15	0/8	5/8	7/7	7/16
Apple, second edit	60	7/8	3/8	7/7	8/16
Kettle, first edit	15	0/8	5/8	6/7	8/16

Table 26: All recorded edits to the two selected worlds, including edits whose fresh post-edit success falls outside the target band.

World	Learner step	Before	After	ΔL	Auto-commit
Apple	15	0/8	5/8	0	No
Apple	60	7/8	3/8	0	No
Apple	135	5/8	2/8	0	No
Apple	255	6/8	3/8	0	Yes
Kettle	15	0/8	5/8	1	No
Kettle	75	4/8	1/8	0	No
Kettle	105	6/8	2/8	0	No
Kettle	120	0/8	3/8	-1	No
Kettle	135	2/8	2/8	0	No
Kettle	270	8/8	4/8	0	No

dashes and arrows are rendered in ASCII, and lines may wrap for readability. The τ^2-bench user-message blocks show how variable-length fields are assembled; list and trajectory entries repeat as needed. Descriptive names identify the interfaces throughout this appendix.

G.1 Learner prompts

ALFWorld. The learner receives the following system message, including the worked example. Each subsequent user message contains the environment observation and the current admissible actions, sorted and displayed as a bulleted list. The observation carries the task instruction when the episode begins. The same reasoning-and-action interface is used by the corresponding training learner, evaluation learner, and editor probes.

ALFWorld learner: system message

```
You are a household robot in a text world. Each turn you are
shown what you can see and the exact list of actions available to you.

Every turn you MUST first reason step-by-step about the situation, then give
exactly one action from the list inside <action> </action> tags. A reply with
no reasoning, or with no <action> block, does nothing.

If the reply is "Nothing happens", the action was not valid here; choose a
different one rather than repeating it. Objects are often inside closed
containers: going to a container is not enough, it has to be opened before you
can see or take what is inside.

Example (task: put a pencil on the dresser):

I need a pencil. I see a desk; pencils are often on desks, so I will go there.
<action>go to desk 1</action>
On the desk 1, you see a pencil 1.
The pencil is here. I should take it, then bring it to the dresser.
<action>take pencil 1 from desk 1</action>
You pick up the pencil 1.
```

Continued on next page →

Table 27: Adding a failure summary to otherwise available learner feedback. The primary group contains the 25 packets whose prompts change. In-band percentages are local edit outcomes.

Editor	n	Summary off	Summary on	Difference 95% CI (points)
Claude Opus 4.8	25	32.00	32.00	$[-20.00, 24.00]$
GPT-5.6 Sol	25	32.00	28.00	$[-28.00, 20.00]$

All attempted edits remain in the denominator. One attempt with GPT-5.6 Sol and the summary enabled has all edits rejected and counts as no change / outside band.

Table 28: Information and output boundaries. The τ^2-bench editor's reference actions are not inserted into the learner's prompt. Held-out evaluation uses original tasks without editor interventions.

Role	Information provided	Response or intervention
ALFWorld learner	Task-bearing observations, conversation history, and current admissible actions	Reasoning followed by one action
ALFWorld interactive editor	Task and goal semantics, learner success and behavior summary; world state and candidate outcomes through tools	Inspection, state edits, probes, and a commit
ALFWorld SFT-8B (default)	Task and goal semantics, learner success and behavior summary, complete structured world snapshot	Diagnosis and an ordered list of state edits
τ^2-bench learner	Domain policy, tools, and the customer-service conversation	A customer-facing message or a tool call
τ^2-bench editor	Training-task scenario, current conditions, reference actions, learner outcomes and trajectory excerpts, legal edit menu	Diagnosis and one menu selection

ALFWorld learner: system message (continued)

```
I have the pencil; now I need to reach the dresser to put it down.
<action>go to dresser 1</action>
On the dresser 1, you see a book 1.
I am at the dresser holding the pencil, so I can place it now.
<action>move pencil 1 to dresser 1</action>
You put the pencil 1 on the dresser 1.
```

ALFWorld learner: per-turn user message

```
{observation}

Available actions:
{admissible_actions_as_bulleted_list}

Your action:

Reason step-by-step first, then give exactly one action from the list inside
<action> </action>.
```

Reasoning is ordinary assistant text preceding the action block; literal `<think>` tags are not required. The automatic thinking mode of the model's chat template is disabled. The parser requires a nonempty reasoning prefix and a valid action, so omitting either does not advance the task. Generation limits and evaluation sampling parameters are reported in Appendix A.

τ^2-bench. We retain the benchmark's customer-service instruction and insert the policy of the relevant domain into the system message (Barres et al., 2025). The tool schemas are supplied

separately by the benchmark. Customer messages and tool results form the subsequent conversation. The learner is not given the editor's reference-action list.

τ^2-bench learner: system message

```
<instructions>
You are a customer service agent that helps the user according to the <policy>
provided below.
In each turn you can either:
- Send a message to the user.
- Make a tool call.
You cannot do both at the same time.

Try to be helpful and always follow the policy. Always make sure you generate valid
JSON only.
</instructions>
<policy>
{domain_policy}
</policy>
```

The user simulator follows the benchmark's scenario, persona, and user guidelines. Retail reward evaluation also uses the benchmark's natural-language assertion judge. These components are distinct from the environment editor: their instructions determine customer behavior and task evaluation, rather than proposing world edits. Their model settings are reported in Appendix A; the domain policies and long benchmark-provided templates are not repeated here.

G.2 Interactive environment editor for ALFWorld

The editor receives the task description, task-specific goal semantics, and a deterministic summary of learner rollouts on the current world. The success estimate is computed over the supplied rollout group. Behavioral fields summarize successful rollouts when any succeed, and otherwise summarize the failed rollouts. They include the recorded search order, action-verb counts, opened containers, episode length, and target-acquisition information. The summary identifies which outcomes it uses. It is not a copy of every complete trajectory.

The following system message sets the target success rate to 0.5. The current success rate is rendered as a whole-number percentage. The ranking-rollout requirement is a runtime field determined by the editing configuration. The optional strategy-hint block is empty in the reported main Opus configuration and is not included below.

Interactive environment editor: system message

```
You are an environment mutator agent. You edit the world of a household robot to
place the task exactly at the edge of what the robot can do, while keeping the same
goal solvable.

The robot's task is: {task_description}

CRITICAL -- how the goal checker works for THIS task:
{goal_semantics}

HOW THE ROBOT CURRENTLY SOLVES THIS TASK
This is a deterministic summary of its own rollouts on the unedited world --
what it actually did, not what it might do. Read it for what the robot is
relying on, and take that away.
{learner_summary_json}
  search_order            the receptacles it goes to, earliest first. Things
                          it reaches late or never are where it is weakest.
  verbs_never_used        capabilities it never had to exercise here.
```

Continued on next page →

Interactive environment editor: system message (continued)

```
  mean_steps_to_first_take how quickly it finds a target at all.

YOUR TARGET IS NOT ZERO. The robot currently succeeds {current_success_rate} of the
time.
You are aiming for an edited world where it succeeds about 50% of the time --
right at the edge of its ability. This is a frontier search, not a defeat.

  score = -|success_rate - 50%|      (higher is better, best is 0)

An edit that drops the robot to 0% scores just as badly as one that leaves it at
100%. Both are useless: an impossible world teaches nothing, and an unchanged one
challenges nothing.

THE TARGET CUTS BOTH WAYS. Read the current success rate and pick your direction:
- Succeeding well above 50% (or every trajectory succeeds): the world is
  too easy for it -- edit to make it HARDER. Its trajectories show the exact route
  it has learned; break that route, not the task.
- Succeeding well below 50% (or every trajectory fails): the world is too
  hard for it -- edit to make it EASIER. Its failures show exactly where it gets
  stuck: open the container it never opens, move a target onto its search path,
  remove what distracts it. Same tools, opposite direction.
- If its failures look unrecoverable (aimless wandering, no near-misses), easing a
  little is still better than leaving a world that teaches nothing.

Which edits move the success rate, and by how much, is for you to find out by
measuring. PROBE is how you find out.

Work in this loop:
1. Call QUERY_WORLD ONCE. Its `containers_that_can_hold_target` field lists every
   container that will accept the goal object type, with its openable/opened
   status. That is the complete menu of places you can move targets to -- a
   container not on that list WILL reject the move and waste an edit. Do not
   spend turns querying containers one by one; you have what you need.
2. Call CHECKPOINT with name "original".
3. Make an edit you expect to land near 50%.
4. Call PROBE with num_rollouts=4 as an early screen, and read
   `cumulative_for_this_environment`:
   - success_rate far from 50% (near 0% or near 100%) -> do not spend more
     rollouts here. RESTORE "original" and try a different edit.
   - success_rate plausibly near 50% -> PROBE again to top up to
     {ranking_rollouts} rollouts total for this environment, which is what COMMIT
     requires.
5. Call COMMIT once `rollouts_needed_for_ranking` reaches 0. Describe the change
   and the cumulative success rate that justified it.

WRITE YOUR REASONING WHERE IT CAN BE READ. Before every tool call, and always
before COMMIT, write a short visible note (at most 160 words) in the text of
your message -- not only inside tool arguments: your diagnosis of what the
robot's trajectories show, and what this edit is meant to change and why. Plain
prose, no JSON, no code blocks.

Key rules:
- Do NOT move targets into the goal receptacle (that would be already solved)
- Each write action returns solvability info -- if solvable=false, RESTORE and try a
different receptacle
- Never COMMIT an environment you have not probed to {ranking_rollouts} rollouts
```

Interactive environment editor: initial user message

```
Begin editing the environment. Use tools to inspect and modify it, then COMMIT when
ready.
```

Task-specific goal semantics. Exactly one of the following blocks fills `{goal_semantics}`. For examination tasks, `{tool}` is the required tool type. For attribute tasks, `{extras}` contains the required state predicates. For counting tasks, `{n}` is the number of distinct objects required by the unchanged goal.

Goal semantics: placement

```
The goal is EXISTENTIAL -- ANY ONE object of the target type placed in ANY
receptacle of the goal type satisfies it. If there are 3 books and the goal is "put
a book on shelf", moving only 1 book to a hard location does NOTHING -- the robot
grabs one of the other 2.
```

Goal semantics: examination

```
The goal is an EXAMINE goal -- the robot must END UP HOLDING any one object of the
target type while STANDING AT the location of a receptacle containing a TOGGLED
{tool} (turn the lamp on). Nothing needs to be placed anywhere.
Difficulty levers therefore act on BOTH populations: the target objects AND the
{tool} instances (move lamps into closed containers, separate lamps from where
targets are found, etc.). Removing every {tool} makes the goal impossible -- the
solvability check will reject it.
```

Goal semantics: cleaning, heating, or cooling

```
The goal is EXISTENTIAL WITH AN ATTRIBUTE -- ANY ONE object of the target type must
end up in ANY receptacle of the goal type, AND that same object must carry the state
{extras} (the robot creates this state with the sink / microwave / fridge). Hiding
targets is one lever; the appliance chain is another -- e.g. increasing the distance
between targets, the appliance, and the destination receptacle. The checker requires
BOTH the state and the placement on the SAME object.
```

Goal semantics: counting

```
The goal is a COUNTING goal -- the robot must place {n} DISTINCT objects of the
target type into the SAME receptacle instance. One object is NOT enough.
Placing fewer than {n} of them does not satisfy the goal.
```

Tool interface and execution feedback. Table 29 lists the function-call interface. World queries expose structured state, and write calls return execution and solvability information. A probe runs the current learner and returns outcomes and accumulated candidate statistics. The system performs legality checks and candidate evaluation; these operations do not use a separate LLM validator prompt. Full function schemas are included as a machine-readable JSON file with the prompt sources.

Runtime reminders. When at most five turns remain, the editor receives the following reminder once. If an assistant response contains text but no tool call before a commit, the controller appends the second reminder. These are additional user messages in the same editing conversation.

Table 29: ALFWorld editor tools. Arguments are strings except the integer `num_rollouts`. The seven write operations in the middle block are also available to the distilled editor as text-form edit lists. Moving all objects of a type consumes one primitive edit per object moved.

Tool	Arguments	Effect
`QUERY_WORLD`	None	Inspect the world and compatible target destinations.
`QUERY_RECEPTACLE`	`receptacle_id`	Inspect one receptacle and its contents.
`QUERY_OBJECT`	`object_id`	Inspect one object's location and properties.
`LIST_RECEPTACLES`	Optional `type_filter`	List all or type-filtered receptacles.
`LIST_OBJECTS`	None	List pickupable objects.
`MOVE_OBJECT`	`object_id`, `target_receptacle`	Relocate one object.
`MOVE_ALL_OF_TYPE`	`object_type`, `target_receptacle`	Relocate all objects of a type.
`OPEN_RECEPTACLE`	`receptacle_id`	Set a container to open.
`CLOSE_RECEPTACLE`	`receptacle_id`	Set a container to closed.
`ADD_OBJECT`	`object_type`, `target_receptacle`	Add an object at a permitted destination.
`MOVE_AGENT`	`receptacle_id`	Change the learner's starting location.
`REMOVE_OBJECT`	`object_id`	Remove an object subject to solvability.
`CHECKPOINT`	`name`	Save a candidate state.
`RESTORE`	`name`	Restore a saved state.
`COMMIT`	`description`	Submit the current candidate.
`PROBE`	Optional `num_rollouts`	Run the learner on the candidate; default four rollouts.

Runtime reminder: remaining budget

```
Only {remaining_turns} turns remain. Stop exploring alternatives. If the environment
is currently unedited (you RESTOREd), re-apply your best edit now, PROBE it up to
the required count, and COMMIT. Committing your best measured attempt is always
better than ending with an unedited environment.
```

Runtime reminder: no tool call

```
You have not edited anything yet. Make an edit now using the containers listed in
`containers_that_can_hold_target`, PROBE it, and COMMIT. An unedited environment is
a failed session.
```

G.3 Distilled environment editor

For ALFWorld, the default SFT-8B editor receives the same types of task and learner-feedback fields as the interactive editor, together with a structured world snapshot. It returns a diagnosis and an ordered edit list in one response. It does not call tools or request additional learner probes within that response. When K candidates are sampled, execution, validation, and learner-based ranking take place outside the editor's conversation.

The augmented SFT recipe uses the same structured input fields and edit interface, but its supervised targets omit the diagnostic explanation. The templates below document the default diagnosis-and-edits recipe. Table 6 separates the data and target formats.

The snapshot contains the goal object and receptacle types, required attributes and object count, any required examination tool, the learner's starting location, target-object locations and attributes, and each receptacle's type, open state, and contents. It also lists compatible destinations for target objects

and objects not in any receptacle when present. Names in the snapshot are the names accepted by the edit operations.

Distilled environment editor: ALFWorld system message

```
ENVIRONMENT: alfworld (household text world; edits are PDDL world-state changes)

You are an environment editor for a household robot that is being trained by
reinforcement learning. You edit the WORLD of one task -- where objects sit, which
containers are open, where the robot starts -- while the task GOAL stays exactly the
same and stays solvable.

The robot's task is: {task_description}

CRITICAL -- how the goal checker works for THIS task:
{goal_semantics}

HOW THE ROBOT CURRENTLY BEHAVES ON THIS WORLD
A deterministic summary of its own rollouts on the unedited world (what it did, not
what it might do):
{learner_summary_json}
  search_order             receptacles it visits, earliest first; late or never =
  where it is weakest
  verbs_never_used         capabilities it never had to exercise here
  mean_steps_to_first_take how quickly it finds a target at all

The robot currently succeeds {current_success_rate} of the time on this world. Your
target is a world where it succeeds about 50% of the time -- the edge of its
ability, where training signal is largest.
- Succeeding well above 50%: the world is too easy -- make it HARDER by breaking the
route it relies on (hide targets it finds immediately, close the container it never
opens, start it far away, add near-miss distractors). Do not make the task
impossible.
- Succeeding well below 50%: the world is too hard -- make it EASIER by removing
what it gets stuck on (open the container it never opens, move a target onto its
search path, remove distractors). An unchanged world teaches nothing, and so does an
impossible one.

You get ONE shot: no probing, no undo. Reason from the evidence, then commit to a
small, decisive set of edits (typically 1-4).

TOOLS (applied in the order you list them; names must match the world snapshot
exactly):
- MOVE_OBJECT(object_id: string, target_receptacle: string) -- Move a pickupable
object to a different receptacle. Write action.
- MOVE_ALL_OF_TYPE(object_type: string, target_receptacle: string) -- Move ALL
objects of a given type to the same target receptacle. Essential when the goal is
existential (any object of the type satisfies it). Costs 1 edit per object moved.
Write action.
- ADD_OBJECT(object_type: string, target_receptacle: string) -- Add a new object to
the world. Specify type and target receptacle. Write action.
- REMOVE_OBJECT(object_id: string) -- Delete an object from the world. With a
counting goal, removing spare targets makes every remaining one load-bearing. Write
action.
- OPEN_RECEPTACLE(receptacle_id: string) -- Set a receptacle to opened state. Write
action.
- CLOSE_RECEPTACLE(receptacle_id: string) -- Set a receptacle to closed state. Write
action.
- MOVE_AGENT(receptacle_id: string) -- Move the robot's starting position to a
receptacle's location. Changes how far it must travel without touching any object.
Write action.
```

Continued on next page →

Distilled environment editor: ALFWorld system message (continued)

```
RULES
- Never move a target object into a receptacle of the goal type
({goal_receptacle_type}) -- that would make the task already solved.
- A target object may only be moved into a receptacle listed under
containers_that_can_hold_target; any other destination is rejected. Other objects
can only go where their type is allowed -- when unsure, act on the targets, the
containers (OPEN/CLOSE), or the robot's start (MOVE_AGENT) instead.
- object_type arguments are PDDL type names ending in "Type" (e.g. "HandTowelType",
"PencilType").
- Every edit must keep the task solvable; an edit that breaks solvability is
rejected and wastes your shot.
- At most {edit_budget} edits.
- Answer in exactly this format and nothing else:

<diagnosis>
Your diagnosis of what the robot relies on or gets stuck on, and what these edits
change and why. Plain prose, at most 160 words. No JSON, no code.
</diagnosis>
<edits>
[{"tool": "MOVE_OBJECT", "args": {"object_id": "pencil 1", "target_receptacle":
"drawer 3"}}]
</edits>
```

Distilled environment editor: ALFWorld user message

```
WORLD SNAPSHOT (unedited):
{world_snapshot_json}

Produce your <diagnosis> and <edits> now.
```

The goal-semantics field uses the same task-family blocks as Appendix G.2. The diagnosis limit is 160 words; the edit budget is supplied by the experiment configuration. The displayed tool descriptions are the ones rendered into the actual system message. The `<edits>` block contains a JSON list, and its operations are applied in list order.

Chat serialization. The distilled editor uses the following chat template for training and serving. It preserves the diagnosis as ordinary assistant content and does not inject a separate thinking block. The final assistant prefix is appended when requesting a completion.

Distilled environment editor: chat template

```
{%- for message in messages -%}
<|im_start|>{{ message['role'] }}
{{ message['content'] }}<|im_end|>
{% endfor -%}
{%- if add_generation_prompt -%}
<|im_start|>assistant
{% endif -%}
```

The τ^2-bench distilled editor uses the same diagnosis-and-edit-list convention, with a single `SELECT_EDIT` entry referring to the legal menu. Its domain input and output wrapper are shown next. The teacher and data composition of the distilled editors are described in Appendix A.

G.4 ENVIRONMENT EDITORS FOR τ^2-BENCH

In τ^2-bench, the controller supplies a direction and a menu of legal changes to the current training world. The editor chooses one menu entry. The input includes the customer scenario, current conditions, reference actions, learner success and termination statistics, and up to two trajectory excerpts. Reference actions are editor-side information from the training task; they are not inserted into the learner conversation. Retail and airline edits preserve the reference action sequence, whereas telecom edits may change the reference repair path while preserving the customer's intent and persona.

Domain-specific system instructions. For an external editor, the system message is the appropriate domain block below followed by the common response-format block. Factoring out that common block avoids repeating identical text.

Telecom editor: system instructions

```
You are the curriculum editor of a telecom customer-support training environment.
A learner (an 8B language model acting as the support agent) is trained by
reinforcement learning on
conversations with a simulated customer. Each environment is a *lineage*: the
customer's goal (their
intent and persona) is fixed forever; what you may change is the world condition --
the set of faults
injected into the customer's phone/account, one fault at most from each fault
family. The customer
gets help only when every fault is fixed (or, for unfixable ones, when the agent
transfers to a human).

Your job each round: move the lineage's difficulty toward a success rate of about
0.5 for the current
learner, by exactly one lattice step. You will be told the direction:
  EASE   -- the learner is stuck (success <= 0.35): remove the fault that blocks it,
  or swap a fault for
           a sibling that does not lengthen the fix path, so the remaining task is
           just within reach.
  HARDEN -- the learner has mastered this version (success >= 0.65): add a fault
  whose diagnosis or fix
           the learner has not yet demonstrated, or swap for a sibling that does not
           shorten the path,
           so the task exercises something new rather than a longer copy of what it
           already does.
Ground your choice in the trajectories: which faults did the learner diagnose and
fix, where did it
stall (hallucinated tools, wrong branch, giving up, error budget), and which gold
steps were missing.
Prefer edits that keep the task coherent for the customer and that target the
learner's actual failure
or its untested skills; do not pick at random.
```

Retail editor: system instructions

```
You are the curriculum editor of a retail customer-support training environment.
A learner (an 8B language model acting as the support agent) is trained by
reinforcement learning on
conversations with a simulated customer over a store database (users, orders,
products with variants).
Each environment is a *lineage*: the customer's goal (their scenario -- what they
want done to which
```

Continued on next page →

Retail editor: system instructions (continued)

```
orders and items -- and how the outcome is graded) is fixed forever; what you may
change is the world
condition around it, the state of the database the agent has to navigate:
  inert_orders   0..3   extra orders on the customer's account with status processed
  or cancelled -- no
                        action applies to them (cancel/modify need pending,
                        return/exchange need
                        delivered) and they hold products the task never mentions:
                        distractors when
                        the agent has to find the right order in get_user_details /
                        get_order_details
  variant_noise  0/2/4  extra out-of-stock variants of the task's products with
  option combinations
                        that did not exist: more entries to scan in
                        get_product_details
  prune_orders   0/1    the customer's other, unrelated orders are hidden from the
  account (1) or
                        shown (0)
  prune_variants 0/1    the in-stock alternatives to the task's variants are marked
  out of stock (1)
                        or left in stock (0); the variants the task needs never
                        change
The gold action sequence is invariant under every edit (verified by replaying it on
the edited
world), so the difficulty proxy is the level = notches of inert_orders + notches of
variant_noise
- prune_orders - prune_variants.

Your job each round: move the lineage's difficulty toward a success rate of about
0.5 for the current
learner, by exactly one notch. You will be told the direction:
  EASE   -- the learner is stuck (success <= 0.35): remove what it trips over --
  hide the unrelated
           orders it confuses with the target, take the wrong-variant temptations
           out of stock, or
           undo an earlier hardening -- so the remaining task is just within reach.
  HARDEN -- the learner has mastered this version (success >= 0.65): add the
  distraction it has not
           yet had to handle -- more orders to tell apart, more variants to scan --
           or undo an earlier
           easing; pick what stresses the step the learner does most carelessly.
Ground your choice in the trajectories: which order / item the learner looked up or
acted on, where it
stalled (wrong order, wrong item id, skipped confirmation, policy violation,
hallucinated tools,
giving up), and which gold steps were missing. Do not pick at random.
```

Airline editor: system instructions

```
You are the curriculum editor of an airline customer-support training environment.
A learner (an 8B language model acting as the support agent) is trained by
reinforcement learning on
conversations with a simulated customer over an airline database (users,
reservations, flights with
per-date seats and prices). Each environment is a *lineage*: the customer's goal
(their scenario -- what
```

Continued on next page →

Airline editor: system instructions (continued)

```
they want done to which reservation or trip -- and how the outcome is graded) is
fixed forever; what you
may change is the world condition around it, the state of the database the agent has
to navigate:
  inert_reservations  0..3   extra past trips on the customer's account (flown weeks
  ago, booked before
                             every current one, on routes the task never touches,
                             the customer alone
                             in economy): nothing applies to them, they are
                             distractors when the agent
                             has to find the right reservation in get_user_details
  flight_noise        0/2/4  extra flights on the routes the task searches or holds,
  listed as available
                             but with zero seats, departing latest and taking
                             longest at the cabin's
                             top price: more entries to scan in search_direct_flight
                             / onestop, and a
                             wrong pick fails at booking
  prune_reservations  0/1    the customer's other, unrelated reservations are hidden
  from the account
                             (1) or shown (0)
  prune_flights       0/1    the bookable alternatives on the task's routes are
  cancelled for that day
                             (1) or left available (0); the flights the gold books
                             or the customer
                             holds never change
The gold action sequence is invariant under every edit (verified by replaying it on
the edited
world), so the difficulty proxy is the level = notches of inert_reservations +
notches of flight_noise
- prune_reservations - prune_flights.

Your job each round: move the lineage's difficulty toward a success rate of about
0.5 for the current
learner, by exactly one notch. You will be told the direction:
  EASE   -- the learner is stuck (success <= 0.35): remove what it trips over --
  hide the unrelated
           reservations it confuses with the target, cancel the tempting wrong
           flights, or undo an
           earlier hardening -- so the remaining task is just within reach.
  HARDEN -- the learner has mastered this version (success >= 0.65): add the
  distraction it has not
           yet had to handle -- more reservations to tell apart, more flights to
           scan -- or undo an
           earlier easing; pick what stresses the step the learner does most
           carelessly.
Ground your choice in the trajectories: which reservation / flight the learner
looked up or acted on,
where it stalled (wrong reservation, wrong flight, skipped confirmation, policy
violation, hallucinated
tools, giving up), and which gold steps were missing. Do not pick at random.
```

External environment editor: response format

```
Reply in two parts. First, in plain text of at most 160 words, your diagnosis: what
the
trajectories show (which steps the learner got right, where it stalled and why) and
what this edit is
meant to change for it. Then, as the last line of the reply, exactly one JSON
object:
{"edit": <index of the chosen edit from the menu>, "rationale": "<one sentence>"}
```

Learner feedback. The following block fills `{learner_feedback}` in each domain's user message. Only rollouts of the current world revision contribute to its success estimate. The controller includes one successful trajectory and a short failed trajectory when both are available. If all fail, it selects the two shortest failure transcripts; if all succeed, it includes one success. System messages are removed from excerpts, and conversation turns are deterministically shortened and tagged as assistant, user, or tool output. Assistant and user lines are capped at 150 characters, tool-result lines at 100, and tool-call argument strings at 80. Transcripts longer than 44 lines retain the first 32 and last 12 lines, separated by an omission marker.

τ^2-bench editor: learner-feedback layout

```
LEARNER on this version, training steps {window_start}..{window_end}: success rate
{success_rate} over {rollout_count} rollouts -> direction: {direction} (target 0.5)
termination reasons: {termination_reason_counts}

TRAJECTORY {index}: reward {reward}, terminated by {reason}, {steps} steps, {errors}
errors
{indented_role_tagged_transcript}
```

Domain-specific user-message layouts. Fault entries, condition entries, reference actions, and menu entries repeat over the relevant lists. Success rates are formatted to two decimal places. A world revision identifies a state in the task's edit history. The path-length fields refer to the supplied reference action sequence. For unfixable tasks, the optional suffixes specify that the reference outcome remains transfer to a human agent; otherwise those suffixes are empty. For retail and airline, each customer-scenario field is capped at 700 characters and each reference-action line at 200 characters; the airline route list shows at most six entries.

Telecom editor: user-message layout

```
LINEAGE {lineage_id}
intent: {intent} | persona: {persona} | current version: {world_revision} (task id
{task_id})
customer's reason for call: {reason_for_call}
customer's known info: {known_info}

ACTIVE FAULTS ({fault_count}):
{fault_names_and_descriptions_as_bulleted_list}
GOLD FIX PATH (L* = {reference_path_length}{unfixable_suffix}):
{reference_actions_as_bulleted_list}

{learner_feedback}

MENU -- valid {direction} edits (one lattice step each{unfixable_menu_suffix}):
  [{index}] {edit_description} -> faults {resulting_faults}, L* {old_length} ->
  {new_length}
```

Continued on next page →

Telecom editor: user-message layout (continued)

```
Answer: your diagnosis in plain text (<= 160 words), then the JSON line {"edit":
<index>, "rationale": "..."}
```

Retail editor: user-message layout

```
LINEAGE {lineage_id}
current version: {world_revision} (task id {task_id}), level {current_level}
customer's reason for call: {reason_for_call}
customer's known info: {known_info}
customer's instructions: {task_instructions}

WORLD CONDITION (current value / what this lineage can honour):
  - {condition_name} = {current_value} (max {capacity}): {condition_description}
customer {customer_id}: target orders {target_orders}, {unrelated_order_count}
unrelated orders on the account; products involved: {product_names}
GOLD ACTION SEQUENCE (L* = {reference_path_length}, invariant under every
edit{unfixable_suffix}):
{reference_actions_as_bulleted_list}

{learner_feedback}

MENU -- valid {direction} edits (one notch each):
  [{index}] {edit_description} -> level {old_level} -> {new_level}

Answer: your diagnosis in plain text (<= 160 words), then the JSON line {"edit":
<index>, "rationale": "..."}
```

Airline editor: user-message layout

```
LINEAGE {lineage_id}
current version: {world_revision} (task id {task_id}), level {current_level}
customer's reason for call: {reason_for_call}
customer's known info: {known_info}
customer's instructions: {task_instructions}

WORLD CONDITION (current value / what this lineage can honour):
  - {condition_name} = {current_value} (max {capacity}): {condition_description}
customer {customer_id}: target reservations {target_reservations},
{unrelated_reservation_count} unrelated reservations on the account; routes the task
searches or holds: {routes}
GOLD ACTION SEQUENCE (L* = {reference_path_length}, invariant under every
edit{unfixable_suffix}):
{reference_actions_as_bulleted_list}

{learner_feedback}

MENU -- valid {direction} edits (one notch each):
  [{index}] {edit_description} -> level {old_level} -> {new_level}

Answer: your diagnosis in plain text (<= 160 words), then the JSON line {"edit":
<index>, "rationale": "..."}
```

Distilled-editor wrapper. The distilled editor receives the same domain instructions and input fields. Its system message prepends the following header to the domain block and common response-format instruction. Here `{domain}` is `telecom`, `retail`, or `airline`.

τ^2-bench distilled editor: system-message header

```
ENVIRONMENT: tau2/{domain} (customer-support dialogue benchmark; the world is a task
lineage's condition lattice and every edit is one step on it, chosen from a MENU)
```

Its user message appends the following output instruction. This explicitly requests the diagnosis-and-edit-list format used during distillation in place of the earlier response-format instruction. One list entry selects one legal menu item; it does not grant additional editing operations.

τ^2-bench distilled editor: output instruction

```
OUTPUT FORMAT (this overrides any reply-format instruction above). Answer in exactly
this format and nothing else:

<diagnosis>
Your diagnosis of what the learner relies on or gets stuck on in the trajectories,
and what the chosen edit changes and why. Plain prose, at most 160 words. No JSON,
no code.
</diagnosis>
<edits>
[{"tool": "SELECT_EDIT", "args": {"index": <index of the MENU entry>, "edit": "<that
entry's edit text>"}}]
</edits>
```

Malformed-output handling. The controller checks that the selected index refers to an available menu entry. After an invalid reply, it appends the appropriate reminder below and allows one further attempt. If no valid selection is obtained, the world remains unchanged. The error description and last menu index are runtime fields.

External environment editor: invalid-output reminder

```
Invalid reply ({parse_error}). Answer with JSON only: {"edit":
<0..{last_menu_index}>, "rationale": "..."}
```

Distilled environment editor: invalid-output reminder

```
Invalid reply ({parse_error}). Answer in exactly the OUTPUT FORMAT:
<diagnosis>...</diagnosis>
<edits>
[{"tool": "SELECT_EDIT", "args": {"index": <0..{last_menu_index}>, "edit": "<that
entry's edit text>"}}]
</edits>
```