TAME: TRANSFER-AWARE MODEL-HARNESS CO-EVOLUTION THROUGH LEARNED HARNESS REVISIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic post-training relies on an execution harness that both assists the model and shapes the trajectories from which it learns. A harness that improves immediate success, however, need not produce capability that transfers when that support changes. We examine this distinction through crossed training- and deployment-harness evaluation, separating same-harness performance, capability retained after support removal, and transfer across a common evaluation family. This evaluation tests for *harness rank reversal*, in which a harness with stronger initial performance yields a weaker trained model under common deployment conditions. This motivates TAME (Transfer-Aware Model-Harness Co-Evolution), which trains a Feedback-Agent language model with reinforcement learning to generate persistent, executable revisions to the training harness. Each revision precedes a fixed-budget inner training stage for the Task-Specific Agent and receives a reward based on the resulting performance change under independently fixed reference harnesses. Specifically, we use token-level proximal policy optimization to train the Feedback-Agent on complete revision-and-training trajectories, assigning credit through transfer gains across subsequent model updates. This makes teaching value an explicit objective for harness adaptation: the capability the model acquires through training, rather than only the success it achieves while supported by the harness.

1 INTRODUCTION

An agent's execution harness controls what its language model observes, which tools it can call, and how failures are handled. During post-training, the same program controls the trajectories that update the model. Documentation may expose a legal action; a planner may supply a subgoal; a parser may repair a malformed command. These interventions improve the supported system, but immediate success does not identify which harness produces the most useful model after training.

This distinction matters whenever the post-trained model will be reused under an harness specified independently of its training harness. Same-harness evaluation combines two effects: what the model learned and what the wrapper continues to supply. We therefore ask a narrower question: *which training harness produces capability that transfers beyond the harness under which it was acquired?* Our crossed evaluation trains matched models under different harnesses and evaluates every endpoint under the same reference harnesses. It separately measures attached performance, capability retained after support removal, and transfer across a deployment family.

We observe *harness rank reversal*. On ALFWorld, the executive harness is strongest before training, yet its model trails models trained with leaner harnesses under a common low-assistance evaluator after 100M tokens, as shown in Figure 1. Harness assistance can expose useful trajectories in some regimes and substitute for behavior the model must learn in others.

Prior systems already edit agent programs or alternate harness and weight updates. SIA (Hebbar et al., 2026) uses a frozen Feedback-Agent to revise programs, while WHALE (Kim et al., 2026) alternates executable harness search with model training. Our focus is the learning target for the reviser. TAME trains a Feedback-Agent from the downstream learning caused by its revisions, measured under fixed deployment harnesses rather than the evolving training program itself.

The resulting problem is sequential, in that a revision changes the next training data, the updated model changes which revision is useful later, and accepted source changes persist across stages.

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.

TAME therefore collects complete model-harness trajectories and updates the program generator only after the batch is complete. Across most experiments and evaluation settings, the learned reviser indeed improves over fixed and frozen-generator controls.

2 RELATED WORK

Reinforcement learning through agent harnesses. Multi-turn agent RL methods such as RAGEN (Wang et al., 2025) and WebAgent-R1 (Wei et al., 2025) train language-model policies from long-horizon environment interaction. Agent Lightning (Luo et al., 2025) separates agent execution from policy optimization, while ClawGym II (Song et al., 2026), Polar (Xu et al., 2026), and OpenForgeRL (Yu et al., 2026) recover training trajectories from complex, opaque, or heterogeneous agent runtimes. Agent Lightning, ClawGym II, Polar, and OpenForgeRL make rich harnesses compatible with model-weight updates without requiring every runtime to be rewritten as a conventional RL environment. The harness itself, however, is generally supplied as a fixed execution infrastructure or as a predefined training condition. TAME instead treats the training harness as an outer decision and evaluates it by the model update it induces.

Optimizing prompts, programs, and harnesses. OPRO (Yang et al., 2023), DSPy (Khattab et al., 2023), and TextGrad (Yuksekgonul et al., 2024) optimize prompts or other components of compound language-model systems from task feedback. Automated Design of Agentic Systems (Hu et al., 2024) and the Darwin Gödel Machine (Zhang et al., 2025) broaden this search to executable agent code. Meta-Harness (Lee et al., 2026) directly searches harness source using candidate code, execution traces, and scores. HarnessDev (Wu et al., 2026) evaluates whether language models can create and iteratively improve runnable harnesses, and whether those harnesses transfer across held-out tasks and runtime models. These works mainly optimize or evaluate the program surrounding a current model; DSPy can also fine-tune module weights. Our question is complementary: after a harness generates training experience, how much capability is stored in the updated model and retained under independently fixed deployment harnesses?

Joint model-harness adaptation. The closest joint-adaptation systems change both model weights and the surrounding program. SIA (Hebbar et al., 2026) uses a language-model Feedback-Agent to choose harness or weight updates from execution feedback. WHALE (Kim et al., 2026) alternates model fine-tuning with Meta-Harness search under the updated model. TAME does not claim alternating model and harness updates as new. Its distinction is the learning target and the learned object: the Feedback-Agent parameters are optimized from delayed post-update transfer gains, with credit propagated across complete sequences of persistent revisions. The frozen-generator and native-system-reward controls isolate this learned revision policy from a strong pretrained reviser and from optimizing the evolving model-harness system itself. Table 1 compares these methods by what they change and what they train. In the methods that edit harness code, the reviser is a pretrained LLM whose parameters stay fixed, whether or not the task model is updated. TAME trains the reviser and scores each revision after the task model has trained under it, on reference harnesses that the revision cannot change.

Learning to teach. Curriculum learning (Bengio et al., 2009), automated curriculum learning (Graves et al., 2017), and unsupervised environment design (Dennis et al., 2020) adapt the examples, tasks, or environments from which a student learns. Learning-to-teach (Fan et al., 2018) methods go further by training a teacher from downstream student performance, including policies over teaching strategies and dynamically generated loss functions (Wu et al., 2018). TAME adopts this teacher-student perspective for agentic post-training, but the teacher's action is an executable interaction program: it changes the information, control flow, parsing, and recovery that produce the student's trajectories. The common-reference evaluation phase then measures the resulting student capability without giving a revision direct credit for assistance that is attached at evaluation time.

3 MEASURING A HARNESS AS A TEACHER

Let π_θ be the task policy, H an execution harness, and $\mathcal{D}$ a task distribution. For $x \sim \mathcal{D}$, a trajectory follows $\tau \sim P(\cdot \mid x, \pi_\theta, H)$ and an immutable verifier V returns the task outcome. Additionally, we

Table 1: Design comparison with the closest methods; scores are omitted because tasks, models, and budgets differ. The task model solves the tasks; the reviser proposes revisions (in Learning to Teach, a teacher that selects training data), and "Reviser trained" refers to the reviser's own parameters. ✓: documented in the source; ×: absent from the described method; —: not applicable. RSFT: rejection-sampling fine-tuning.

Method	**Harness code edits**	**Task-model update**	**Reviser trained**	**Signal for a revision**
Agent Lightning (Luo et al., 2025)	×	RL	—	—
Meta-Harness (Lee et al., 2026)	✓	None	×	Candidate score and traces
SIA (Hebbar et al., 2026)	✓	LoRA RL	×	Traces and task metrics
WHALE (Kim et al., 2026)	✓	RSFT	×	Candidate score and traces, current weights
Learning to Teach (Fan et al., 2018)	—	SGD / Adam	✓	Student dev-set speed or accuracy
TAME (ours)	✓	GRPO	✓	Post-update gain on ref. harnesses

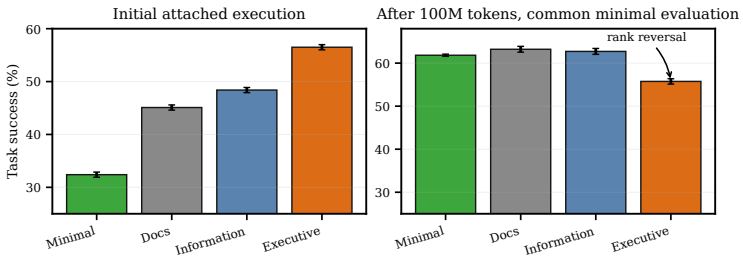

Figure 1: Harness rank reversal on ALFWorld with Qwen3-4B. The left panel reports initial task success when each harness is attached to the same starting model. The right panel reports performance after 100M training tokens under the corresponding harness, with every resulting model evaluated through the common minimal harness $H_{\min}$. The reversal shows that the harness with the strongest initial execution does not produce the strongest model under common evaluation.

define the execution value

$$J_{\mathcal{D}}(\theta, H) = \mathbb{E}_{x,\tau}[V(x,\tau)], \tag{1}$$

which measures the complete model-harness system. However, it does not, by itself, separate the capability stored in the model from the support supplied by H.

We have all training runs start from the same state ζ_0, with $\theta_0 = \theta(\zeta_0)$. Then, training under protocol c_i for budget B produces $\zeta_i = \textsc{Update}_B(\zeta_0, c_i, \mathcal{D}_{\text{inner}})$ and $\theta_i = \theta(\zeta_i)$. We distinguish the initial attached value $Z_i = J(\theta_0, c_i)$, the attached endpoint $S_i = J(\theta_i, c_i)$, the de-scaffolded gain

$$A_i = J_{\mathcal{D}}(\theta_i, H_{\min}) - J_{\mathcal{D}}(\theta_0, H_{\min}), \tag{2}$$

and the crossed gain $X_{i\to j} = J_{\mathcal{D}}(\theta_i, H_j) - J_{\mathcal{D}}(\theta_0, H_j)$. A de-scaffolded rank reversal occurs when $Z_i > Z_j$ but $A_i < A_j$. This means that the initially stronger system is the weaker teacher under the common low-assistance evaluator, which is the inequality set that we are mainly interested in.

The minimal harness provides a controlled view of capability after substantial support is removed, but it represents only one possible deployment condition. We therefore also evaluate each model through a fixed family of reference harnesses, denoted $\mathcal{H}_{\text{score}}$. Complete description can be found in Table 7. For weights $\mathbf{w} \in \Delta(\mathcal{H}_{\text{score}})$, the aggregate transfer score is

$$\mathcal{T}_{\mathcal{D}}^{\mathbf{w}}(\theta) = \sum_{H \in \mathcal{H}_{\text{score}}} w_H \, J_{\mathcal{D}}(\theta, H). \tag{3}$$

The evaluation family and its weights are fixed across training protocols, so a method cannot improve this score merely by making its own evaluation harness more supportive. In the main setting, we assign weight $1/2$ to $H_{\min}$ and weight $1/8$ to each of H_{docs}, H_{info}, H_{exec}, and H_{null}. We report the component scores and the aggregate to expose tradeoffs among deployment conditions. A separate held-out harness, excluded from Feedback-Agent training and model selection, tests whether the

Table 2: Minimal-minus-executive success after 100M training tokens under the common minimal evaluator. Positive values favor minimal-harness training.

Task model / domain	Difference (pp)	Ordering
Qwen3-4B / ALFWorld	$+6.1 \pm 0.4$	reversed
Qwen3-4B / ScienceWorld	-2.4 ± 0.8	executive retained
Qwen3-4B / WebShop	$+4.0 \pm 0.5$	reversed
Qwen3-4B / τ^2 Telecom	-6.1 ± 1.7	executive retained
Llama-3.2-3B / ALFWorld	$+6.6 \pm 1.1$	reversed
Qwen2.5-3B / ALFWorld	$+6.5 \pm 1.6$	reversed

learned improvement extends beyond the harnesses used to define the outer objective. As in Table 2, rank reversal depends on task domains and cannot be explained by richer or leaner assistance alone.

4 TRANSFER-AWARE HARNESS REVISION

The study above suggests to us that harness adaptation should indeed optimize the learning it induces, rather than only the execution quality of the currently supported system. TAME therefore treats an executable harness revision as an outer action that precedes each fixed-budget task-model update. This ordering lets the Feedback-Agent receive credit for how its revision changes the model's subsequent learning and adaptation process. Figure 2 shows one stage of this process. GRPO updates the Task-Specific Agent in every stage, whereas token-level PPO updates the Feedback-Agent and its critic only between batches of complete revision-and-training trajectories; the harness changes only through source patches.

Executable revisions before model updates. We adapt an off-the-shelf program-revision harness and associated definitions from SIA (Hebbar et al., 2026), in which the Feedback-Agent observes the current executable source A_g, selected interaction traces τ_g, learning feedback $\mathcal{E}_g$, and the task specification $\mathcal{U}$. At stage g, it samples a source-code patch from μ_ϕ. A fixed validator then either applies the patch or retains the current harness:

$$A_{g+1} = \mathcal{F}_\phi(A_g, \tau_g, \mathcal{E}_g, \mathcal{U}), \qquad \zeta_{g+1} = \textsc{Update}_{B_g}(\zeta_g, A_{g+1}, \mathcal{D}_{\text{inner}}). \tag{4}$$

The resulting harness A_{g+1} generates the trajectories used by the next inner update, which trains the Task-Specific Agent for budget B_g. A valid revision becomes the active harness and persists into later stages. If a patch is invalid or reaches the generation limit, the system simply keeps A_g and still performs the scheduled model update without any attempts of modification. Importantly, the rejected patch remains part of the sampled outer trajectory, which allows the Feedback-Agent to learn from its resulting transfer outcome rather than filtering unsuccessful proposals.

More specifically, the editable surface typically includes prompt construction, context management, observation summarization, parsing, and bounded retry logic. We in particular fix the task definition, the common verifier, and the suite of reference evaluation harnesses. This separation allows the revision policy to change how training experience is produced without changing the task or the criterion used to evaluate what the model had learned.

Post-update transfer reward. A harness revision should receive credit for the model update it induces, rather than for making its own evaluation harness more supportive. We therefore evaluate the task model before and after each inner training stage using the same tasks, reference harnesses, and rollout seeds. The stage reward is the resulting change in common transfer:

$$r_g^{\text{out}} = \mathcal{T}^{\mathbf{w}}_{\mathcal{D}_{\text{meta}}}(\theta_{g+1}) - \mathcal{T}^{\mathbf{w}}_{\mathcal{D}_{\text{meta}}}(\theta_g). \tag{5}$$

The pre-update score at stage $g+1$ is the post-update score from stage g. With discount $\gamma = 1$, the stage rewards therefore telescope as follows,

$$\sum_{g=0}^{K-1} r_g^{\text{out}} = \mathcal{T}^{\mathbf{w}}_{\mathcal{D}_{\text{meta}}}(\theta_K) - \mathcal{T}^{\mathbf{w}}_{\mathcal{D}_{\text{meta}}}(\theta_0).$$

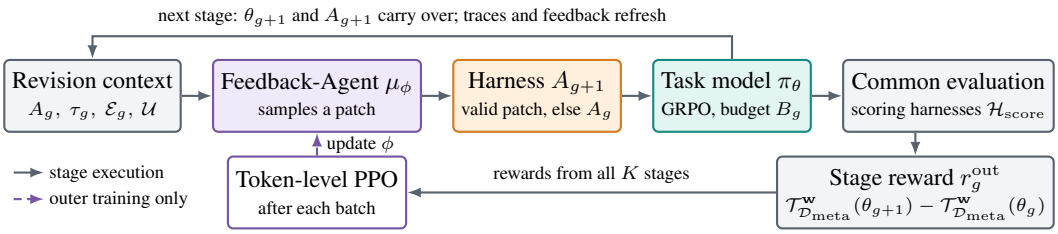

Figure 2: Overview of TAME. A fixed validator applies the Feedback-Agent's patch if valid. The resulting harness produces rollouts for one fixed-budget GRPO update of the task model, whose score change under the fixed reference harnesses is the stage reward. Token-level PPO updates the Feedback-Agent between batches of complete K-stage trajectories; at deployment, ϕ^* is frozen while revision and training continue.

Algorithm 1 TAME: Co-evolution with a transfer-aware Feedback-Agent

Require: Validation-selected ϕ^*; fresh $(\zeta_0, A_0, \mathcal{P}_0)$;
task specification $\mathcal{U}$, fixed revision harness and budgets $(B_g)_{g=0}^{K-1}$;
permitted development feedback and a final-test suite.

Freeze ϕ^* obtained with Algorithms 2 and 3, evaluation programs, weights, and test-selection rules.
$(\tau_0, \mathcal{E}_0) \leftarrow \textsc{InitialFeedback}(\zeta_0, A_0, \mathcal{P}_0)$
for $g = 0, \ldots, K-1$ **do**
  $A_{g+1} \leftarrow \mathcal{F}_{\phi^*}(A_g, \tau_g, \mathcal{E}_g, \mathcal{U})$
  $\zeta_{g+1} \leftarrow \textsc{Update}_{B_g}(\zeta_g, A_{g+1}, \mathcal{D}_{\text{inner}})$
  $(\tau_{g+1}, \mathcal{E}_{g+1}, \mathcal{P}_{g+1}) \leftarrow \textsc{RefreshFeedback}(\zeta_{g+1}, A_{g+1}, \mathcal{P}_g)$
end for
$\theta_K \leftarrow \theta(\zeta_K)$
$\mathcal{R}_{\text{test}} \leftarrow \textsc{EvaluateFinal}(\theta_K, \mathcal{D}_{\text{test}}, \mathcal{H}_{\text{eval}})$
return $\theta_K, A_K, \mathcal{P}_K, \mathcal{R}_{\text{test}}$

A positive reward thus means that the model improved under the common evaluation family during that stage. A limitation to this credit assignment scheme is that it does not, by itself, identify the causal effect of the revision, since continued training under the unchanged harness may also improve the model. Seeing this, we consider estimating that effect separately through a matched KEEP continuation from the same parent learning state.

Credit across revisions. Each revision is a program-valued action represented by a sequence of sampled tokens. We concatenate the revision tokens from all K stages of an outer trajectory and index them by n. Let u_n denote the stage context and revision prefix preceding token y_n. The token reward ρ_n is zero for interior tokens and equals r_g^{out} at the final sampled token of revision g. We note in particular that a revision boundary is not an outer terminal. When another stage remains, the successor is the next revision context after task-model training and evaluation.

Let $\bar{v}_n$ be the frozen critic prediction at prefix u_n, and let m_n equal one whenever the outer trajectory continues and zero only at its true terminal state. We compute generalized advantages (Schulman et al., 2016) backward through the complete trajectory:

$$\delta_n = \rho_n + \gamma m_n \bar{v}_{n+1} - \bar{v}_n,$$
$$\widehat{A}_n = \delta_n + \gamma\lambda m_n \widehat{A}_{n+1}, \qquad \widehat{R}_n = \widehat{A}_n + \bar{v}_n. \tag{6}$$

Our default setting uses $\gamma = \lambda = 1$. On a complete trajectory, this reduces $\widehat{A}_n$ to the sum of all remaining transfer gains minus the prefix-value baseline. Earlier revisions can therefore receive credit for learning that appears only after later task-model updates.

Token-level PPO. Let ϕ_{old} denote the Feedback-Agent snapshot that generated the current batch. For each sampled token, we form the likelihood ratio $q_n(\phi) = \frac{\mu_\phi(y_n|u_n)}{\mu_{\phi_{\text{old}}}(y_n|u_n)}$. Given N complete outer

trajectories, the Feedback-Agent maximizes the clipped token-level objective

$$L_{\mathrm{FA}}(\phi) = \frac{1}{N}\sum_{i=1}^{N}\sum_{n=1}^{T_i} \min\left\{ q_n^{(i)}(\phi)\widehat{A}_n^{(i)}, \mathrm{clip}\left(q_n^{(i)}(\phi), 1-\epsilon, 1+\epsilon\right)\widehat{A}_n^{(i)} \right\}, \tag{7}$$

while the critic minimizes

$$L_V(\psi) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{n=1}^{T_i}\left(v_\psi(u_n^{(i)}) - \widehat{R}_n^{(i)}\right)^2. \tag{8}$$

Both objectives sum token contributions within each trajectory and average over trajectories, rather than normalizing each revision by its length. The behavior-policy probabilities, advantages, and value targets remain fixed throughout the PPO epochs. We then discard the batch and collect fresh trajectories with the updated Feedback-Agent. Thus, the Feedback-Agent is updated between complete collection batches instead of between successive task-model stages within a trajectory. Appendix A gives the full collection, optimization, and frozen-deployment procedures.

5 Algorithmic Design

Deploying the learned revision policy. Outer training produces a Feedback-Agent that has learned which executable harness revisions tend to improve subsequent model learning. We select its checkpoint using the held-out development split and then evaluate that revision policy on fresh task-model training runs. The deployment run thus tests whether the learned revision rule transfers to a new model-training trajectory, rather than continuing to optimize the Feedback-Agent on the trajectory being evaluated.

Algorithm 1 summarizes this procedure. At stage g, ζ_g denotes the complete learning state of the task-model, A_g the active executable harness, and $\mathcal{P}_g$ their revision and training history. The fixed Feedback-Agent ϕ^* receives the current source, allowed execution records τ_g, development feedback $\mathcal{E}_g$, and task specification $\mathcal{U}$. It then immediately proposes the next harness A_{g+1}, after which the Task-Specific Agent undergoes one fixed-budget inner update under that harness. The resulting model, program, and feedback are carried on to the next stage.

The ordering in Algorithm 1 is essential: each revision precedes the model update whose learning outcome it is intended to improve, and both the updated model state and the active program persist into later stages. Freezing ϕ^* therefore does not freeze the harness sequence. The task-model parameters change through Update, whereas the harness changes through the state-conditioned revisions of the fixed policy as the model, source, and observed feedback evolve.

6 Experimental Design

Our experiments address three questions. First, does the harness that gives the strongest immediate execution also produce the strongest model after training? Second, does reinforcement learning actually improve an otherwise identical program reviser? Third, does optimizing common-reference transfer produce different deployment behavior from optimizing the evolving model-harness system? Our study addresses the first question with fixed training protocols and cross-harness evaluation. Our main study addresses the remaining questions through revision-policy comparisons and ablation studies. We evaluate the resulting task models through the same reference harnesses and report their attached-system performance separately, distinguishing model capability under common deployment conditions from performance with each method's own final program.

Models, tasks, and training budgets. The primary setting trains Qwen3-4B-Instruct-2507 (Yang et al., 2025) on ALFWorld (Shridhar et al., 2021). A separate code-capable language model serves as the Feedback-Agent. GRPO (Shao et al., 2024) updates the Task-Specific Agent, whereas token-level PPO (Schulman et al., 2017) updates the harness-revision policy. Within each matched comparison, we hold the initial task-model learning state, inner optimizer, training-token budget, and evaluation procedure fixed. Matched revision-policy arms also share the starting harness.

Each primary deployment consists of four stages with 25M all-model-call training tokens per stage, for 100M tokens along the retained training trajectory. Every call introduced by the training harness,

including retries and auxiliary model calls, counts against this budget. Sensitivity evaluations use 25M, 50M, 200M, 400M, and 800M tokens. ScienceWorld (Wang et al., 2022) provides a domain replication, while WebShop (Yao et al., 2022) and τ^2-bench Telecom (Barres et al., 2025) broaden the task family. Llama-3.2-3B and Qwen2.5-3B test dependence on the task-model family. We cross all three task models with these four domains, yielding twelve model-domain settings. WebArena-Verified (Zhou et al., 2024; El hattami et al., 2025) is used only for final evaluation and never contributes revision feedback or model selection. Long-horizon experiments keep the Feedback-Agent frozen after a 4-stage training and extend fresh deployments to 8, 16, and 32 stages.

Task separation and common evaluation. Disjoint task sets support inner task-model training, Feedback-Agent reward construction, hyperparameter and checkpoint selection, and final testing. Related task variants remain in the same split. In our main experiment setting, the same fixed reference family and weights in Equation 3 define transfer on the outer-reward and final-test splits:

$$\mathcal{H}_{\text{score}} = \{H_{\text{min}}, H_{\text{docs}}, H_{\text{info}}, H_{\text{exec}}, H_{\text{null}}\}, \qquad \mathbf{w} = (1/2, 1/8, 1/8, 1/8, 1/8).$$

We report the aggregate score, minimal-harness performance, and every component score separately. Two additional harnesses, H_{plan} and H_{retry}, provide diagnostic evaluations. Each endpoint therefore includes eight fixed-harness scores and the final attached-system score. The latter uses the method's own final program and is distinct from performance under the shared H_{exec} evaluator.

Crossed harness evaluation. Each of the seven named harnesses defines a fixed training protocol. We evaluate every resulting model under all eight fixed harnesses at each declared budget, producing a cross-harness matrix for each model-domain setting. We compare five rankings spanning initial attached execution, final attached execution, minimal-harness performance, weighted common-reference performance, and held-out-harness performance. Specifically, the primary executive-versus-minimal reversal case requires an initial executive advantage followed by an advantage for minimal-harness training under the common minimal evaluator. We report all twelve settings, including those in which this ordering does not reverse.

Baselines. The validation-selected fixed teacher repeats one protocol throughout training. A fixed information-executive mixture provides a second simple control. Our one-shot program-optimization baseline adapts Meta-Harness (Lee et al., 2026): it searches for a training program before fresh task-model training and then keeps the selected program fixed. The frozen-generator baseline adapts SIA (Hebbar et al., 2026) and uses the same pretrained Feedback-Agent, context packing, editable source harness, decoding distribution, validity checks, and persistent revision schedule as TAME. It retains its initial generator parameters, whereas TAME uses token-level PPO to update them between batches of complete revision-and-training trajectories. Their generator parameters are frozen during fresh deployments while task-model training and harness revision continue.

We also adapt WHALE (Kim et al., 2026) as an alternating execution-search baseline. It ranks candidate harnesses using the current task model with its weights held fixed, then trains under the selected program. The HarnessDev (Wu et al., 2026) adaptation evolves a harness with the runtime model fixed and subsequently uses the resulting program as a fixed teacher for fresh task-model training. These competitive baselines have a common inner learner and editable harness. All evaluated candidates and continuations, including discarded ones, contribute to search cost.

7 RESULTS

We first examine whether immediate execution performance predicts teaching value. We then separate the benefit of learning the reviser from the choice of its reward, evaluate the role of persistent revisions and credit across training stages, and study how the conclusions change with deployment conditions.

7.1 THE BEST EXECUTOR IS NOT ALWAYS THE BEST TEACHER

The executive harness improves initial attached performance relative to the minimal harness in every evaluated setting, yet its advantage does not consistently survive training; see Figure 3. At the primary 100M-token endpoint, minimal-harness training outperforms executive-harness training on Qwen3-4B/ALFWorld when both models are evaluated through the same minimal harness. The reversal also

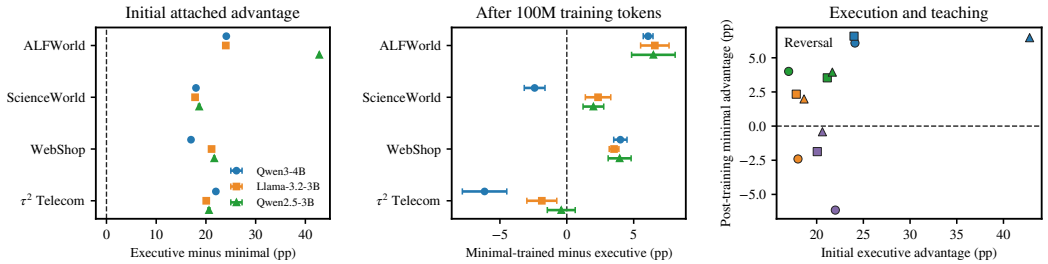

Figure 3: Execution quality and teaching value across twelve settings. Positive values in each panel identify rank reversal.

occurs for Llama-3.2 and Qwen2.5 on ALFWorld and for all three models on WebShop. Qwen3-4B on ScienceWorld and all Telecom settings instead retain an executive advantage. Immediate assistance and teaching value are therefore distinct criteria, and neither richer nor leaner assistance is uniformly the better training choice.

Table 8 quantifies the consequence of choosing a teacher by initial execution performance. It compares the model produced by that choice with the best common-reference endpoint among the seven fixed teachers. It also identifies settings where an informational teacher is preferable even though minimal-harness training does not outperform executive training. The appendix provides the complete crossed matrices and rankings, showing how these comparisons depend on the evaluation harness.

7.2 Learning the reviser changes downstream model performance

We next distinguish the benefit of revising a harness from the benefit of training the reviser. With Qwen3-4B and 100M training tokens, TAME improves common transfer over the validation-selected fixed teacher on ALFWorld and ScienceWorld. We also observe improvement relative to the otherwise identical frozen reviser. Specifically, this ablation setting holds the generator initialization, revision procedure, and task-model training budget fixed, isolating the contribution of learning the revision policy. Figure 4 reports these differences for each model-domain setting. The gains are not uniform: on Qwen3-4B/Telecom, TAME scores lower in common transfer than the frozen generator. Learning the reviser therefore improves the models produced in some settings while remaining sensitive to the domain and task model.

The lower row of Figure 4 isolates the reward objective, with both variants receiving feedback after task-model updates: common-reference reward favors performance across the shared evaluation harnesses, whereas native-system reward favors the final model operating with its own evolved harness. Keeping post-update evaluation in both variants separates the choice of learning objective from the benefit of waiting until after training to assess a revision. Optimizing the final attached system and optimizing performance across deployment harnesses can therefore produce different teachers.

7.3 Persistent revisions and credit across stages improve transfer

Figure 5 tests the main design choices at the same task-model training budget. On Qwen3-4B/ALFWorld, replacing the common-transfer reward with immediate execution reward after the update lowers the common transfer score. Restricting each revision's credit to the next model update, or resetting the source before every revision, also lowers it. Together, these comparisons support evaluating revisions through subsequent model learning, allowing useful edits to persist, and assigning credit across successive training stages.

Appendix F.6 reports the minimal- and held-out-harness scores separately, distinguishing changes that extend beyond the primary aggregate from those concentrated in one evaluation harness.

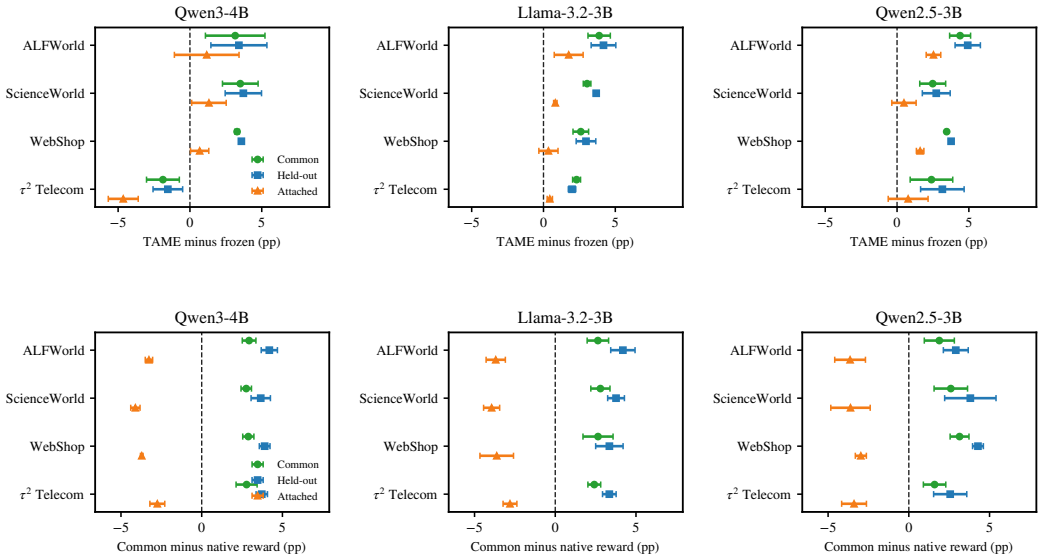

Figure 4: Learning the reviser and choosing its reward. Top row: TAME minus the otherwise identical frozen Feedback-Agent. Bottom row: common-reference PPO minus native-harness PPO. Appendix F.5 reports within- and between-fit variation for the common, minimal, and held-out measures.

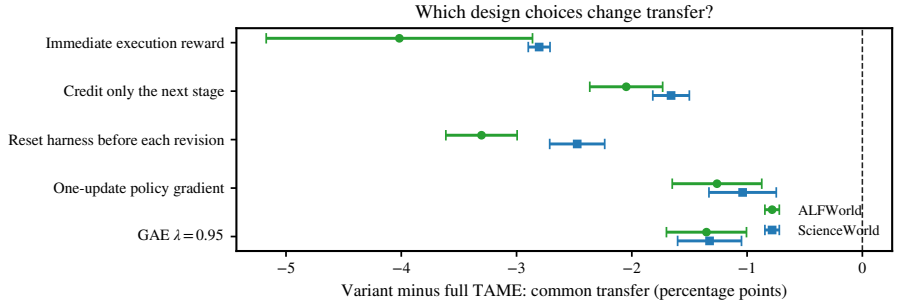

Figure 5: Design ablations for Qwen3-4B after 100M training tokens. Each point is a paired difference from full TAME; negative values favor TAME. Bars show the standard deviation across three fit-level means. The policy-gradient variant changes both the clipped update and reuse of collected trajectories, so this comparison does not isolate clipping alone. Appendix F.6 gives all twelve settings.

8 Conclusion

Choosing or editing a harness for agentic post-training in model-harness co-evolution differs from choosing one for immediate execution. Our crossed evaluation shows that initial harness rankings can change or reverse after training, depending on the task model and domain. TAME makes this a sequential decision-making objective: a Feedback-Agent generates persistent, executable harness revisions and learns from the transfer gains of later model updates, with credit assigned across revision sequences. In the primary ALFWorld and ScienceWorld settings, the learned Feedback-Agent improves common transfer over fixed teachers and the otherwise identical frozen generator. Among post-update rewards, common-reference reward favors shared deployment harnesses, and native-system reward favors the final attached system. These benefits are setting-dependent: executive assistance retains its value in some domains, and the frozen reviser is stronger in others. Harness adaptation should match the model's intended use; for models deployed beyond their training harness, the quality of assistance should be judged by the capability it helps the model acquire and retain.

AI USE STATEMENT

Generative AI tools, including ChatGPT and Claude Code, were used to refine hypotheses and experimental methodology, assist implementation and debugging, search and summarize literature, draft and edit text. The authors take responsibility for the complete submission.

REPRODUCIBILITY STATEMENT

Sections 4 and 6 specify the optimization target, update ordering, principal comparisons, and information boundaries. Appendices A, B, and C provide the complete algorithms and hyperparameters.

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 *International Conference on Machine Learning*, 2009.

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*, 2020.

Amine El hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. WebArena Verified: Reliable evaluation for web agents. In *Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=94tlGxmqkN.

Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. *arXiv preprint arXiv:1805.03643*, 2018. URL https://arxiv.org/abs/1805.03643.

Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *International Conference on Machine Learning*, 2017.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: Self improving AI with harness & weight updates. *arXiv preprint arXiv:2605.27276*, 2026. URL https://arxiv.org/abs/2605.27276.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. URL https://arxiv.org/abs/2408.08435.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023. URL https://arxiv.org/abs/2310.03714.

Haechan Kim, Yoonho Lee, Gisang Lee, Chelsea Finn, and Kangwook Lee. WHALE: A simple recipe for joint harness-weight optimization. *arXiv preprint arXiv:2609.00196*, 2026. URL https://arxiv.org/abs/2609.00196.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026. URL https://arxiv.org/abs/2603.28052.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent Lightning: Train ANY AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025. URL https://arxiv.org/abs/2508.03680.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, et al. 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.

Huatong Song et al. ClawGym II: Exploring black-box RL on agent harness. *arXiv preprint arXiv:2608.16798*, 2026. URL https://arxiv.org/abs/2608.16798.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Conference on Empirical Methods in Natural Language Processing*, 2022.

Zihan Wang et al. 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.

Zhepei Wei et al. WebAgent-R1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*, 2025. URL https://arxiv.org/abs/2505.16421.

Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jian-Huang Lai, and Tie-Yan Liu. Learning to teach with dynamic loss functions. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Xinping Lei, Qingshui Gu, Yuxuan Zhang, Zexuan Wang, Chen He, Chen Huang, Maojia Song, Zhiyuan Zeng, Shaowen Wang, Jinkai Liu, Yunfeng Shi, Jiaheng Liu, Shen Yan, Wenhao Huang, Ge Zhang, and Wenxuan Zhang. HarnessDev: Can LLMs create and evolve their own agent harness? *arXiv preprint arXiv:2609.01437*, 2026. URL https://arxiv.org/abs/2609.01437.

Binfeng Xu et al. Polar: Agentic RL on any harness at scale. *arXiv preprint arXiv:2605.24220*, 2026. URL https://arxiv.org/abs/2605.24220.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023. URL https://arxiv.org/abs/2309.03409.

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

Xiao Yu et al. OpenForgeRL: Train harness-native agents in any environment. *arXiv preprint arXiv:2607.21557*, 2026. URL https://arxiv.org/abs/2607.21557.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025. URL https://arxiv.org/abs/2505.22954.

Shuyan Zhou et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

A Complete Training and Deployment Procedure

An outer trajectory follows a task model through a sequence of harness revisions and training stages. Each stage generates a program revision, updates the task model under the resulting harness, and evaluates the updated model through the common reference harnesses. The task-model learning state and active program carry forward from one stage to the next. The Feedback-Agent and critic remain fixed throughout collection of a complete batch; their parameters are updated only after that batch has been collected.

At stage g, let ζ_g denote the restorable task-model learning state, A_g the executable harness, and $\mathcal{P}_g$ the revision and feedback history. We use A_g for the harness also denoted by h_g. The joint state and the context visible to the Feedback-Agent are

$$\xi_g = (\zeta_g, A_g, \mathcal{P}_g), \qquad \theta_g = \theta(\zeta_g), \qquad s_g = f(\xi_g). \tag{9}$$

The fixed context constructor f selects the permitted source, interaction traces, and development feedback. This summary need not expose the full learning state, and we do not assume that it is Markov. Several separately sampled continuations may start from the same saved state; all such continuations remain in the same data split. Below, $\bar{\phi} = \phi_{\text{old}}$ and $\bar{\psi} = \psi_{\text{old}}$ denote the generator and critic snapshots used to collect the batch.

A.1 Collecting Revision Trajectories

Algorithm 2 collects one complete outer trajectory. The Feedback-Agent samples a patch from the current harness and permitted feedback. A proposal that terminates within the token limit and passes the fixed validator is applied to the active source. Otherwise, the current harness is retained. In either case, the scheduled inner training stage proceeds, and the proposal remains in the collected data with its original tokens and likelihoods. Its reward is the transfer gain from the resulting model update, including when that update uses the unchanged harness.

Each model checkpoint is evaluated on the same development tasks, reference harnesses, and rollout seeds. Evaluation leaves both the task model and the Feedback-Agent unchanged. Writing the checkpoint score as $z_g = \widehat{\mathcal{T}}^{\mathbf{w}}_{\mathcal{D}_{\text{meta}}}(\theta_g)$, we assign $\widehat{r}^{\text{out}}_g = z_{g+1} - z_g$. The endpoint score is cached and reused as the next stage's baseline, so the recorded rewards satisfy

$$\sum_{g=0}^{K-1} \widehat{r}^{\text{out}}_g = z_K - z_0.$$

The updated learning state, active source, and refreshed feedback then supply the next revision context. This preserves the effects of earlier revisions on both the model and the program inherited by later stages.

A.2 Token-Level Credit and PPO Updates

The Feedback-Agent selects a program revision by generating its tokens. To assign credit across revisions, we index all sampled revision tokens in a complete outer trajectory by $n = 1, \ldots, T$, where $T = \sum_{g=0}^{K-1} L_g$. For token y_n, the conditioning input u_n contains the recorded stage context and the preceding tokens of that revision. This indexing connects the return calculation across stages; each likelihood is still evaluated under its own recorded context. Only Feedback-Agent tokens contribute to its policy loss. Task-model generations, tool responses, and evaluation outcomes provide context or rewards rather than additional Feedback-Agent actions.

The transfer gain from stage g is attached to the last token actually sampled for its revision. All earlier tokens in that revision receive zero immediate reward. Within a revision, the successor context extends the current prefix by one token. At a revision boundary, the successor is the first-token context of the next revision, constructed after the intervening task-model update and evaluation. A sampled termination token therefore ends a revision, while only the end of the complete outer trajectory terminates credit propagation. Table 3 summarizes these cases.

Let $\bar{v}_n = v_{\bar{\psi}}(u_n)$ be the frozen prefix-value estimate. The continuation mask is $m_n = 1$ whenever another generated token remains in the outer trajectory and $m_T = 0$ at its terminal token. In

Algorithm 2 CollectRevisions: collect a complete revision-and-training trajectory. Generator and critic snapshots remain fixed. Every proposal receives the transfer gain from its ensuing model update, including proposals that leave the harness unchanged.

Require: Starting state ξ_0; frozen generator $\bar{\phi}$ and critic $\bar{\psi}$;
task specification $\mathcal{U}$; fixed editing rules and validator;
budgets $(B_g)_{g=0}^{K-1}$; development evaluator on $\mathcal{D}_{\mathrm{meta}}$;
fixed reference harnesses, weights, evaluation tasks, and rollout seeds.

$(\zeta_0, A_0, \mathcal{P}_0) \leftarrow \textsc{Restore}(\xi_0)$; $\mathcal{D} \leftarrow \varnothing$
$z_0 \leftarrow \widehat{\mathcal{T}}^{\mathbf{w}}_{\mathcal{D}_{\mathrm{meta}}}(\theta(\zeta_0))$
$(\tau_0, \mathcal{E}_0) \leftarrow \textsc{InitialFeedback}(\zeta_0, A_0, \mathcal{P}_0)$
for $g = 0, \dots, K-1$ **do**
  $s_g \leftarrow \textsc{Pack}(A_g, \tau_g, \mathcal{E}_g, \mathcal{U})$
  $a_g = (a_{g,1}, \dots, a_{g,L_g}) \sim \mu_{\bar{\phi}}(\cdot \mid s_g)$
  $\ell_{g,t} \leftarrow \log \mu_{\bar{\phi}}(a_{g,t} \mid s_g, a_{g,<t})$, $t = 1, \dots, L_g$
  $\bar{v}_{g,t} \leftarrow v_{\bar{\psi}}(s_g, a_{g,<t})$, $t = 1, \dots, L_g$
  if a_g terminates normally within the token limit and passes the validator **then**
    $A_{g+1} \leftarrow \mathrm{Apply}(A_g, a_g)$
  else
    $A_{g+1} \leftarrow A_g$ ▷ retain the proposal in $\mathcal{D}$
  end if
  $\zeta_{g+1} \leftarrow \textsc{Update}_{B_g}(\zeta_g, A_{g+1}, \mathcal{D}_{\mathrm{inner}})$
  $z_{g+1} \leftarrow \widehat{\mathcal{T}}^{\mathbf{w}}_{\mathcal{D}_{\mathrm{meta}}}(\theta(\zeta_{g+1}))$
  $\widehat{r}^{\mathrm{out}}_g \leftarrow z_{g+1} - z_g$
  Append $(s_g, a_g, \ell_g, \bar{v}_g, \widehat{r}^{\mathrm{out}}_g)$ and the validation outcome to $\mathcal{D}$.
  $(\tau_{g+1}, \mathcal{E}_{g+1}, \mathcal{P}_{g+1}) \leftarrow \textsc{RefreshFeedback}(\zeta_{g+1}, A_{g+1}, \mathcal{P}_g)$
  Cache z_{g+1} as the next baseline; retain the complete successor state.
end for
Index the $T = \sum_{g=0}^{K-1} L_g$ sampled tokens in generation order.
Preserve each token's recorded context and its actual successor context.
Set $m_n \leftarrow 1$ for $n < T$ and $m_T \leftarrow 0$.
return $\mathcal{D}$ with token likelihoods, frozen values, rewards, and continuation masks.

Table 3: Reward placement and value successors for Feedback-Agent tokens. Each stage gain is assigned to the revision's final sampled token.

Token position	Immediate reward	Successor for value estimation
Interior token	0	Current revision with its extended prefix
Final token, another stage remains	$\widehat{r}^{\mathrm{out}}_g$	Next revision context after task-model training and evaluation
Final token of the final stage	$\widehat{r}^{\mathrm{out}}_{K-1}$	Outer terminal state, with value zero
Last token before forced truncation	Transfer gain from training under the unchanged harness	Next revision context if another stage remains; otherwise the outer terminal state

particular, m_n remains one across a revision boundary when a later stage follows. With discount $\gamma = 1$, generalized advantage estimation computes

$$\delta_n = \rho_n + m_n \bar{v}_{n+1} - \bar{v}_n,$$
$$\widehat{A}_n = \delta_n + \lambda m_n \widehat{A}_{n+1}, \qquad \widehat{R}_n = \widehat{A}_n + \bar{v}_n, \tag{10}$$

backward through the complete trajectory, with $\bar{v}_{T+1} = \widehat{A}_{T+1} = 0$. For the primary setting $\lambda = 1$, the intermediate value terms cancel, giving

$$\widehat{A}_n = \sum_{j=n}^{T} \rho_j - \bar{v}_n, \qquad \widehat{R}_n = \sum_{j=n}^{T} \rho_j.$$

Thus, an early revision receives credit for subsequent transfer gains along the collected training trajectory, including gains that arise after later model updates. Values of $\lambda < 1$ place greater reliance

Algorithm 3 TOKENPPO: update the Feedback-Agent and critic from complete outer trajectories. Advantages propagate across revision boundaries, and all targets remain fixed throughout the optimization epochs.

Require: Complete batch $\mathcal{B} = \{\mathcal{D}_i\}_{i=1}^{N}$; collection snapshots $\bar{\phi}, \bar{\psi}$;
discount $\gamma = 1$; GAE parameter λ; clipping threshold ϵ;
learning rates η_ϕ, η_ψ; epoch count E; uniformly sampled trajectory minibatches.

$(\phi, \psi) \leftarrow (\bar{\phi}, \bar{\psi})$
for each $\mathcal{D}_i \in \mathcal{B}$ **do**
  Index its recorded tokens as $(u_n, y_n, \ell_n, \bar{v}_n, m_n)_{n=1}^{T_i}$.
  $\rho_n \leftarrow 0$ for $n = 1, \ldots, T_i$
  for each revision g with final sampled-token index b_g **do**
    $\rho_{b_g} \leftarrow \widehat{r}_g^{\,\mathrm{out}}$
  end for
  $\bar{v}_{T_i+1} \leftarrow 0$; $\widehat{A}_{T_i+1} \leftarrow 0$
  for $n = T_i, T_i - 1, \ldots, 1$ **do**
    $\delta_n \leftarrow \rho_n + m_n \bar{v}_{n+1} - \bar{v}_n$
    $\widehat{A}_n \leftarrow \delta_n + \lambda m_n \widehat{A}_{n+1}$
    $\widehat{R}_n \leftarrow \widehat{A}_n + \bar{v}_n$
  end for
end for
Hold all ℓ_n, $\widehat{A}_n$, and $\widehat{R}_n$ fixed for the following updates.
for $e = 1, \ldots, E$ **do**
  for each trajectory minibatch $\mathcal{M} \subseteq \mathcal{B}$ **do**
    $q_n^{(i)}(\phi) \leftarrow \exp\left(\log \mu_\phi(y_n^{(i)} \mid u_n^{(i)}) - \ell_n^{(i)}\right)$ for tokens in $\mathcal{M}$
    $\phi \leftarrow \phi + \eta_\phi \nabla_\phi L_{\mathrm{FA}}^{\mathcal{M}}(\phi)$ ▷ Equation 7
    $\psi \leftarrow \psi - \eta_\psi \nabla_\psi L_V^{\mathcal{M}}(\psi)$ ▷ Equation 8
  end for
end for
Discard $\mathcal{B}$ before collecting fresh trajectories.
return (ϕ, ψ)

on the critic and attenuate later contributions by token distance. We evaluate this choice separately in the estimator-sensitivity comparisons.

Sampling and token likelihoods. The generator samples from a softmax distribution at a fixed temperature. A fixed, prefix-dependent grammar mask may restrict syntax. Both the behavior and updated-policy likelihoods use this same temperature and masking rule, without top-k or nucleus truncation. The stored log probability therefore describes the distribution that actually generated each token. When a termination token is sampled, it is included in the action and its likelihood record. Forced truncation ends the proposal at the last sampled token, without appending a termination token. The capped proposal is rejected, the scheduled inner update uses the unchanged harness, and its transfer gain is assigned to that last token.

Generator and critic optimization. For each recorded token, PPO compares its likelihood under the current generator with its likelihood under the behavior snapshot

$$q_n(\phi) = \frac{\mu_\phi(y_n|u_n)}{\mu_{\bar{\phi}}(y_n|u_n)}. \tag{11}$$

Both probabilities condition on the same recorded prefix u_n. The clipped generator objective and value-regression objective are given in Section 4. For each minibatch $\mathcal{M}$, they sum token contributions within each trajectory and divide by the number of trajectories $|\mathcal{M}|$. Normalizing each revision by its own length would instead change the relative weighting of the sampled programs.

Algorithm 3 updates the generator and critic only after the complete collection batch is available. Rejected proposals and proposals followed by poor learning outcomes remain in that batch with their original token records and returns. The critic estimates remaining transfer gain from the visible token context; it is not used to differentiate through the task-model optimizer. PPO updates the probability of generated revisions rather than differentiating through program execution or inner training.

A.3 Deploying the Learned Revision Policy

Fresh deployment evaluates the learned revision policy on a new task-model training run. In the main comparison, validation selects the Feedback-Agent checkpoint and deployment configuration. The reward-only comparison uses the final outer checkpoint for both reward variants. Once the checkpoint is chosen, the Feedback-Agent parameters remain fixed throughout deployment. Its outputs can still change with the active harness and refreshed feedback, allowing it to adapt the training program as the task model learns.

Each deployment stage follows the same revision, validation, inner-update, and feedback sequence used during collection. An invalid or capped proposal retains the active harness, and the scheduled task-model update still proceeds. A source revision does not roll back the task-model learning state: the updated model and active program both carry forward to the next stage. Deployment ends after the declared training horizon and budgets, and the resulting model is evaluated on the final-test suite. Final-test outcomes enter neither revision generation nor checkpoint selection, and no Feedback-Agent PPO updates occur during this fresh run.

B Executable Harness Revisions

A harness revision changes the program that produces training experience for the task model. The revision rules specify which parts of that program may change, what information the Feedback-Agent can use, and which parts of the learning and evaluation procedure remain fixed. This section describes how feedback is assembled, how generated patches are checked, and how successive revisions are carried through a training run.

B.1 Feedback Construction and Generation Rules

At each stage, a fixed context constructor assembles four blocks in order: the task specification and editing rules, the current source A_g, selected interaction records τ_g, and development feedback $\mathcal{E}_g$. Each interaction record retains the observation, raw model response, executed action, tool return, and identities of the model and harness that produced it. The feedback block contains the cached common-development score, training statistics, revision-validity outcomes, revision history, and remaining budget. A fixed truncation rule limits the input length while preserving the editing rules and source identity. Missing observations remain absent rather than being reconstructed from other information.

InitialFeedback assembles the permitted records associated with the starting state. It reuses the cached common-development score when available and evaluates the starting model otherwise. RefreshFeedback extends $\mathcal{P}_g$ with the sampled patch, validation outcome, resulting source, task-model update, common-development score, and resource use. It then selects the non-test traces and metrics for the next revision. Records collected at different points in training retain their original model identities, so responses from earlier actor versions remain distinguishable from those of the current checkpoint.

The learned Feedback-Agent and the frozen-generator baseline receive the same generation instruction. It specifies the permitted edit and the learning outcome used to evaluate it:

```
Read the current source, task specification, permitted
execution records, and development feedback.  Return one
patch within the editable portion of the harness, or an
empty patch to keep the current source.  Preserve the task,
verifier, external action semantics, optimizer, evaluation
programs, and resource limits.  Do not read hidden state or
final-test data.  Your patch will precede the next fixed
model-training stage; its learning signal is the resulting
common-transfer gain and not an immediate execution score.
```

The program action consists of the sampled patch tokens, with their likelihoods recorded under the generating policy. An improvement report, when retained, is identified separately as diagnostic text rather than silently included in the action sequence. This distinction keeps the PPO objective tied to the tokens whose sampling probabilities were recorded.

Table 4: Scope of executable harness revision. All harness-generation methods use the same permitted observations, tools, external action semantics, and resource limits. Revisions change how the training harness uses these resources, while the task, inner learner, and reference evaluation remain fixed.

Component	Permitted changes and fixed constraints
Observation and prompt construction	May reorganize declared observations and documentation. Access to hidden environment state and evaluation answers is prohibited.
Context and memory	May maintain task-local summaries derived from permitted observations and interaction history, within the fixed context allowance.
Action parsing and recovery	May revise the declared parsing and retry procedures. Raw model responses and executed external actions are recorded separately.
Tool interaction	May invoke declared tools within fixed call and retry limits. The meaning and effects of external actions remain defined by the environment.
Task definition and verifier	The task specification, environment transitions, legal action space, and verifier remain fixed.
Learning and evaluation	The inner optimizer, training budgets, data splits, reference evaluation harnesses, and scoring weights remain fixed.

B.2 What the Feedback-Agent Can Change

The harness mediates interaction between the task model and its environment. Within the declared editable code, the Feedback-Agent can generate a patch or replace a permitted module. Revisions may reorganize observations and documentation, change context construction or task-local memory, and compose parsing and recovery procedures. The generated program can combine these behaviors without reproducing any of the named reference harnesses.

Table 4 separates the editable behavior from the fixed task, learning procedure, and evaluation rules. Every candidate must pass static checks and bounded execution tests before activation. These checks assess conformance to the editing rules; teaching value is measured through the subsequent task-model update and common evaluation.

The following example changes only context construction. Both versions retain the task and current observation. The replacement also includes up to two recent tool returns and asks the model to check its command format before acting. This is a change to the executable program that prepares the model's input, instead of simply a choice among predefined harness labels.

```
# Current harness module
def make_context(task, observation, recent):
    return task + "\nObservation:\n" + observation

# Example replacement
def make_context(task, observation, recent):
    feedback = "\n".join(
        item["tool_result"] for item in recent[-2:])
    return (task + "\nObservation:\n" + observation
            + "\nRecent tool feedback:\n" + feedback
            + "\nCheck the command format before acting.")
```

A higher success rate while executing with this revised context would measure its immediate benefit to the supported model. In TAME, the stage reward instead measures the change in common transfer after the model trains under the revised program. The same code change can therefore receive different rewards at different model checkpoints. Conditioning on the current source and observed learning feedback allows the reviser to respond to this dependence on the model's learning state.

B.3 Program Persistence and Revision History

An accepted revision becomes the active harness for the next training stage and remains in use until a later accepted revision changes it. The revision history links each source version to its parent program, sampled patch, validation outcome, and associated training feedback. These records allow the program sequence to be reconstructed and distinguish a new source version from a proposal that retains or reproduces an existing one. Restoring an earlier source version changes the harness used for subsequent training while preserving the task-model learning state reached so far.

The archive records the programs encountered during a run. Importantly, it does not restrict the Feedback-Agent to selecting one of them. Each new action is another generated revision. For example, the Feedback-Agent may reuse earlier code, combine functions from the permitted history, or introduce new code within the same editing rules.

C Training, Evaluation, and Reproducibility Details

Table 5: Model and inner-learning settings. Each task-model comparison keeps its initial checkpoint, trainable parameters, and inner optimizer fixed across harness-revision methods. Tokenizers and chat templates are taken from the same checkpoint as the corresponding model.

Setting	Configuration
Qwen3-4B-Instruct-2507	2507 instruction-tuned release; native tokenizer and non-thinking chat template; full-parameter updates, ∼4.0B parameters, BF16.
Llama-3.2-3B-Instruct	Llama 3.2 instruction-tuned release; native tokenizer and chat template; full-parameter updates, ∼3.21B parameters, BF16.
Qwen2.5-3B-Instruct	Qwen2.5 instruction-tuned release; native tokenizer and chat template; full-parameter updates, ∼3.09B parameters, BF16.
DeepSeek-V2-Lite-Chat	V2-Lite Chat SFT release; native tokenizer and chat template; full-parameter updates, ∼16B total parameters (2.4B activated per token), BF16.
GRPO rollout group and batch sizes	8 trajectories per task group. ALFWorld, WebShop, and InterCode-Bash: 32 groups per batch (256 trajectories). ScienceWorld and τ^2-bench Telecom: 16 groups (128 trajectories). Minibatches of 8 groups; one optimization pass per fresh batch.
GRPO learning rate, clipping, KL terms	AdamW with constant learning rate 1×10^{-6}; ratio clipping $\epsilon_{\text{in}} = 0.2$; KL coefficient $\beta_{\text{KL}} = 0.01$ against the frozen initial task policy; gradient-norm clipping at 1.0. Optimizer state persists across revision stages.
Task-model context, response limits	8,192 total tokens and at most 512 new tokens per call; 16,384 and 1,024, resp., for Telecom and WebArena-Verified. Sampling temperature: 1.0 during training, 0.2 during evaluation; top-$p = 1$, with top-k filtering disabled.
Common evaluation rollouts per task	4 rollouts per task-harness pair, averaged rather than best-of-four.
Long-horizon model–benchmark cells	Qwen3-4B on ALFWorld and ScienceWorld; Llama-3.2-3B on ALFWorld. 8, 16, or 32 stages of 25M all-model-call tokens (200M, 400M, or 800M total). The four-stage-trained Feedback-Agent remains frozen.

Table 6: Task-model and Feedback-Agent configuration.

Setting	Value
Feedback-Agent	Qwen2.5-Coder-7B-Instruct; frozen BF16 backbone; LoRA on attention and MLP projections: $r = 32$, $\alpha_{\text{LoRA}} = 64$, dropout 0.
Critic	Separate copy of the same checkpoint with a zero-initialized scalar prefix-value head; no shared trainable parameters.
Primary horizon and budget	4 stages during training and deployment; 25M tokens per stage, including all training-harness model calls.
Discount and GAE	$\gamma = \lambda = 1$.
Outer collection	8 rounds of 4 complete trajectories; 3.2B inner-training tokens.
Validation	Rounds 2, 4, 6, and 8; two fresh 100M-token runs per checkpoint (0.8B tokens total); select by mean validation transfer.
PPO	Clipping $\epsilon = 0.2$; 4 epochs per batch; 2 complete trajectories per minibatch.
Optimizer	AdamW; constant learning rates 10^{-5} (actor) and 5×10^{-5} (critic); gradient-norm limit 1.0.
Revision generation	One proposal per stage; at most 2,048 new tokens; temperature 0.8, top-$p = 1$, top-k disabled.
Feedback-Agent context	16,384 total tokens; source and permitted records only.
Search budget	4 candidates per search stage; 16 one-shot candidates with one 25M-token trial each. Rank by the development objective; ties favor the earlier proposal.
Replication	3 independent fits per setting, each with 5 paired deployments.
Long-horizon deployment	Frozen generator; 8, 16, or 32 stages (200M, 400M, or 800M tokens).
Cost amortization	1 and 20 deployments; divide shared construction cost by the deployment count.

Token losses are summed within trajectories and averaged across trajectories; each batch is discarded after its PPO epochs. Native tools allow at most one formatting retry per action, with no automatic replay of state-changing calls.

D Trajectory and Intervention

Harness changes can improve execution by informing the model or by altering what is executed on its behalf. The following examples distinguish these mechanisms. A recorded experimental trace additionally identifies the program version, task-model checkpoint, raw response, executed action, and each model and tool call. The common transfer evaluation determines whether any resulting training benefit persists under a different reference harness.

D.1 Information Versus Action Replacement

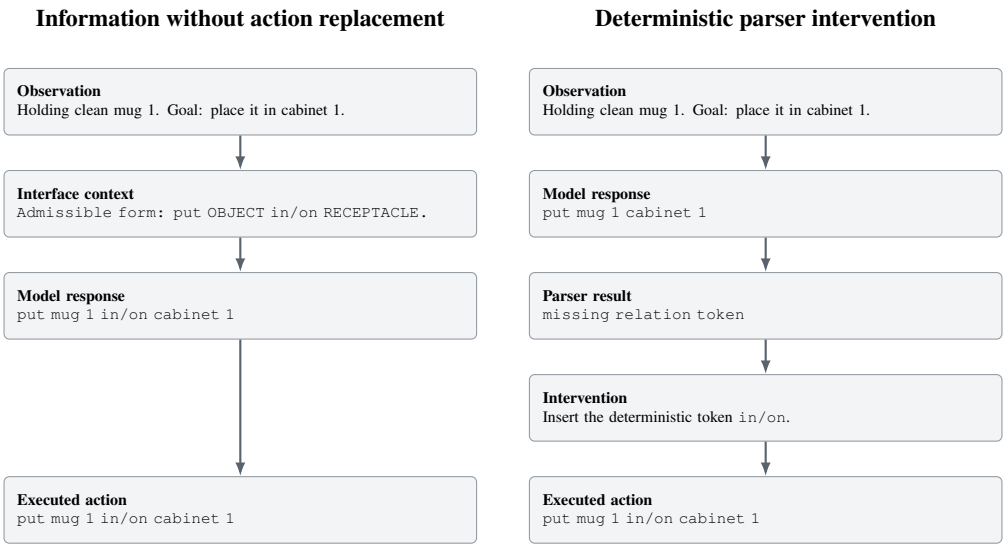

Figure 6: ALFWorld interaction trace distinguishing unchanged model commands from deterministic parser intervention. Raw responses and executed commands identify which component supplied the correction.

D.2 Retry and Tool-Call Attribution

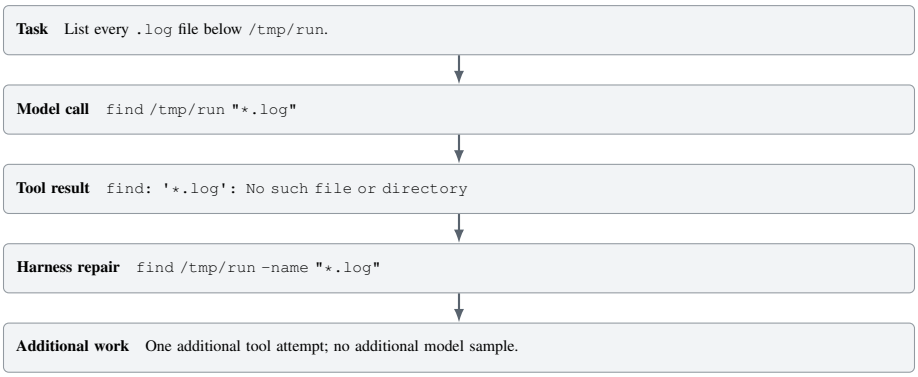

Figure 7: Command-repair example. With no literal file named `*.log` in the working directory, the quoted pattern is interpreted as a second starting path and produces the shown missing-path error. Adding `-name` makes it a filename predicate. The repair adds one tool call and no model sample.

D.3 Benchmark Revision and Interaction Examples

The examples below connect harness changes to the interactions they shape and the learning outcomes used to assess them. They distinguish the model's responses, the actions executed by the harness, and the feedback returned by the environment. Task-level outcomes are reported separately from the post-update transfer gain that trains the Feedback-Agent.

The ALFWorld (Shridhar et al., 2021) trace follows an invalid command and its subsequent correction, making clear that the corrected action is generated by the task model. The ScienceWorld (Wang et al., 2022) trace distinguishes the native cumulative score from the reward received at each step. Its binary success criterion requires full task completion.

The τ^2-bench Telecom (Barres et al., 2025) example preserves the separation between the service agent's tools and the user's device tools. The complete record includes both participants, whereas the service agent receives device-tool results only when the user communicates them. This separation allows the full interaction to be inspected while making clear which information was available to the agent at each decision.

WebArena-Verified (El hattami et al., 2025) is reserved for final testing, whose outcomes do not guide harness revision or checkpoint selection. The final ALFWorld example examines a context revision that passes validation yet produces a negative stage transfer gain. Retaining this failure case makes the distinction between program validity and teaching value concrete: an admissible edit can still lead to a model update that reduces common-transfer performance.

ALFWorld Observed feedback without silent action replacement
Block 2, deployment 1, stage 1
Task. Put a clean mug in cabinet 1. Initially visible: countertop 1, sinkbasin 1, and cabinet 1.

Linked revision and model update. A0 $\rightarrow$ A2; one 25M-token inner stage. Development transfer: $52.27 \rightarrow 61.96\%$; gain $+9.69$ pp; outer reward $+0.0969$. Matched Keep: 61.67%; edit-minus-Keep: $+0.29$ pp.

Revision fragment.

```
def make_context(task, observation, recent):
  errors = [e for e in recent if e.get("invalid", False)]
  return {"task": task, "observation": observation,
      "last_error": errors[-1:],
      "instruction": "Write one complete command."}
```

1. Model. `go to countertop 1`
A dirty mug 1 is on countertop 1. *Score 0; step reward 0.*

2. Model. `take mug 1 from countertop 1`
You pick up mug 1. *Score 0; step reward 0.*

3. Model. `go to sinkbasin 1`
You arrive at sinkbasin 1. *Score 0; step reward 0.*

4. Model. `clean mug 1 with sinkbasin 1`
Mug 1 is clean and remains in inventory. *Score 0; step reward 0.*

5. Model. `go to cabinet 1`
Cabinet 1 is closed. *Score 0; step reward 0.*

6. Model. `open cabinet 1`
Cabinet 1 is open. *Score 0; step reward 0.*

7. Model. `put mug 1 cabinet 1`
Parser: missing relation. No environment action. *Score 0; step reward 0.*

8. Model. `put mug 1 in/on cabinet 1`
The clean mug is in cabinet 1; goal satisfied. *Score 1; step reward 1.*

Attribution. The rejected command remains model behavior. Its correction is a new model sample, not a command inserted by the harness.
Excerpt: 8 model calls and 7 external tool actions.

Figure 8: ALFWorld revision and interaction excerpt.

ScienceWorld Preserving full object referents
Block 2, deployment 2, stage 1
Task. Measure the temperature of the water in the metal pot. The excerpt begins in the room containing the objects.

Linked revision and model update. A1 $\rightarrow$ A2; one 25M-token inner stage. Development transfer: $29.69 \rightarrow 38.24\%$; gain $+8.56$ pp; outer reward $+0.0856$. Matched Keep: 35.69%; edit-minus-Keep: $+2.55$ pp.

Revision fragment.

```
def visible_referents(observed_objects):
  return [item["referent"] for item in observed_objects]

def make_context(task, observation, objects):
  return {"task": task, "observation": observation,
      "referents": visible_referents(objects)}
```

1. Model. `look around`
Visible: a thermometer, water in a metal pot, and water in a jug. *Score 0; step reward 0.*

2. Model. `take thermometer`
The thermometer is in inventory. *Score 0; step reward 0.*

3. Model. `focus on thermometer`
The instrument has been identified. *Score 25; step reward 25.*

4. Model. `use thermometer on water`
Ambiguous target: water in metal pot, or water in jug? *Score 25; step reward 0.*

5. Model. `use thermometer on water in metal pot`
Temperature: 23 degrees Celsius. Requested measurement complete. *Score 100; step reward 75.*

Attribution. Full observed names resolve ambiguity. Native cumulative score and step reward are separate. Aggregate success is one only at full completion.
Excerpt: 5 model calls and 5 external tool actions.

Figure 9: ScienceWorld revision and interaction excerpt.

tau2-bench Telecom Agent-side evidence and user-side actions
Block 2, deployment 2, stage 1
Task. Restore mobile data without changing the plan or billing. Phone-local settings are controlled by the user.

Linked revision and model update. A1 $\rightarrow$ A2; one 25M-token inner stage. Development transfer: $33.16 \rightarrow 39.38\%$; gain $+6.22$ pp; outer reward $+0.0622$. Matched Keep: 37.01%; edit-minus-Keep: $+2.36$ pp.

Revision fragment.

```
def visible_context(messages):
  visible = [m for m in messages if m["role"] != "user_tool"]
  return {"messages": visible,
      "instruction": "Ask the user to check; confirm the result."}
```

1. User. `My mobile internet stopped working. Customer ID: C1001.`
Message delivered to the service agent.

2. Agent. `get_customer_by_id(customer_id="C1001")`
Customer record identifies line L1001; fields abbreviated.

3. Agent. `get_details_by_id(id="L1001")`
Line L1001 is active; account-side service is available.

4. Agent. `Please check whether airplane mode is on.`
Instruction delivered to the user.

5. User. `check_network_status()`
Airplane Mode: ON; Mobile Data Enabled: Yes; other fields omitted. *User-side tool return; not visible to the agent.*

6. User. `Airplane mode is on.`
User reports the device observation to the agent.

7. Agent. `Turn it off, then check whether the phone reconnects.`
Instruction only; the agent does not invoke the user tool.

8. User. `toggle_airplane_mode()`
Airplane mode is OFF. The phone reconnects. *User-side tool return; not visible to the agent.*

9. User. `run_speed_test()`
Speed Test Result: 44.00 Mbps (Good). *User-side tool return; not visible to the agent.*

10. User. `It reconnects and the speed test completes.`
Outcome reported to the agent.

11. Agent. `Connectivity is restored. No plan or billing change is needed.`
Final response. Overall task outcome: success.

Attribution. Both participants are visible here. User-tool returns are hidden from the service agent and reach it only through user messages.
Excerpt: 11 model calls and 5 external tool actions. Agent/user split: 5/6 model calls; 2/3 tool calls.

Figure 10: tau2-bench Telecom revision and interaction excerpt.

WebArena-Verified Testing with a stale browser target
Block 2, deployment 2
Task. Report the status of order 1042. Initially visible: [12] Orders. This is a read-only endpoint audit.

Fixed interface fragment.
```
def validate_target(action, observed_ids):
  target = action.get("element_id")
  if target is not None and target not in observed_ids:
    return {"execute": False, "feedback": "Read the latest view."}
  return {"execute": True, "action": action}
```

1. Model. `click [12]`
Orders page: [31] Search; [18] View order 1042; order 1041 also listed.

2. Model. `type [31] [1042]`
Updated view: [45] View order 1042.

3. Model. `click [18]`
Fixed-interface check: stale element ID. No browser click.

4. Model. `click [45]`
Order 1042 details: Processing. Captured read identifies the same order.

5. Model. `stop [{"order_id":"1042","status":"Processing"}]`
Typed response matches the authored target and captured entity.

Attribution. Excerpt: 5 model calls and 3 external tool actions. Authored response/entity checks both pass; this is not output from the released verifier.

Figure 11: WebArena-Verified revision and interaction excerpt.

ALFWorld A valid revision that discards necessary state
Block 1, deployment 1, stage 2
Task. Place mug 1 in cabinet 1. An earlier observation placed the mug on countertop 2; the revised summarizer keeps only the latest observation.

Linked revision and model update. A2 $\rightarrow$ A3; one 25M-token inner stage. Development transfer: $62.44 \rightarrow 57.86\%$; gain -4.57 pp; outer reward -0.0457. Matched Keep: 60.56%; edit-minus-Keep: -2.70 pp.

Revision fragment.
```
def summarize_recent(recent):
  # Failing candidate: unresolved earlier state is discarded.
  return recent[-1:]
```

1. Model. `go to cabinet 1`
Cabinet 1 is open; it contains no mug. *Score 0; step reward 0.*

2. Model. `put mug 1 in/on cabinet 1`
Failed: mug 1 is not in inventory. *Score 0; step reward 0.*

3. Model. `look`
Cabinet 1 is open. You see nothing inside. *Score 0; step reward 0.*

4. Model. `put mug 1 in/on cabinet 1`
Repeated failure. Episode budget exhausted. *Score 0; step reward 0.*

Attribution. Source validity does not imply teaching value. The failure and its negative stage gain remain in the data; the following stage still updates model weights.
Excerpt: 4 model calls and 4 external tool actions.

Figure 12: ALFWorld revision and interaction excerpt.

E Additional Results

E.1 Harness Rank Reversal

Table 7: Named reference harnesses for characterization and fixed-protocol comparisons. All preserve the task, verifier, model, environment, legal action space, and external action semantics.

Harness	Supplied support	Capability left to the model
H_{min}	Basic observation, strict parser, and minimal action loop	Plan, select, format, and recover without extra support
H_{docs}	Static environment and command documentation	Author every plan and action using clearer harness knowledge
H_{info}	Documentation, admissible actions, and inventory or state summaries	Choose actions with substantial state and harness assistance
H_{plan}	Template-generated subgoal content	Execute a decomposition partly supplied by the protocol
H_{retry}	Deterministic malformed-action repair and bounded retry	Propose actions while some weak proposals receive recovery
H_{exec}	Subgoals, repair, state support, and bounded retry	Operate with the complete executive configuration
H_{null}	The planning harness with empty subgoal content	Control for harness and formatting changes

Table 8: Cost of choosing a teacher by initial execution at 100M training tokens. Initial attached success selects H_{exec} in every setting. The listed teacher has the highest common score among the seven fixed training choices.

Domain	Best teacher	Common loss	Held-out difference
Qwen3-4B			
ALFWorld	H_{docs}	7.03 ± 0.19	$+7.27 \pm 0.28$
ScienceWorld	H_{info}	3.93 ± 0.76	$+4.39 \pm 0.84$
WebShop	H_{info}	5.11 ± 0.38	$+5.31 \pm 0.28$
τ^2 Telecom	H_{exec}	0.00 ± 0.00	$+0.00 \pm 0.00$
Llama-3.2-3B			
ALFWorld	H_{docs}	7.74 ± 1.64	$+8.05 \pm 1.61$
ScienceWorld	H_{info}	3.86 ± 0.44	$+4.03 \pm 0.64$
WebShop	H_{info}	5.97 ± 0.44	$+6.34 \pm 0.31$
τ^2 Telecom	H_{info}	0.48 ± 1.09	$+0.84 \pm 1.04$
Qwen2.5-3B			
ALFWorld	H_{docs}	7.92 ± 2.11	$+8.12 \pm 2.01$
ScienceWorld	H_{info}	3.68 ± 0.84	$+4.00 \pm 0.85$
WebShop	H_{info}	6.05 ± 0.39	$+5.73 \pm 0.63$
τ^2 Telecom	H_{info}	2.15 ± 0.30	$+1.94 \pm 0.57$

E.2 Learning Executable Harness Revisions

These evaluations separate improvement in the Feedback-Agent's own training objective from transfer produced in fresh task-model runs.

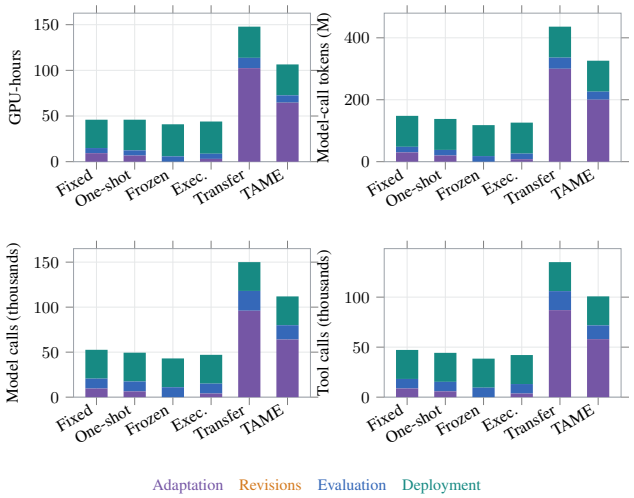

Figure 13: Phase-specific cost for the 100M-token ALFWorld experiment, with shared construction cost amortized over 20 deployments.

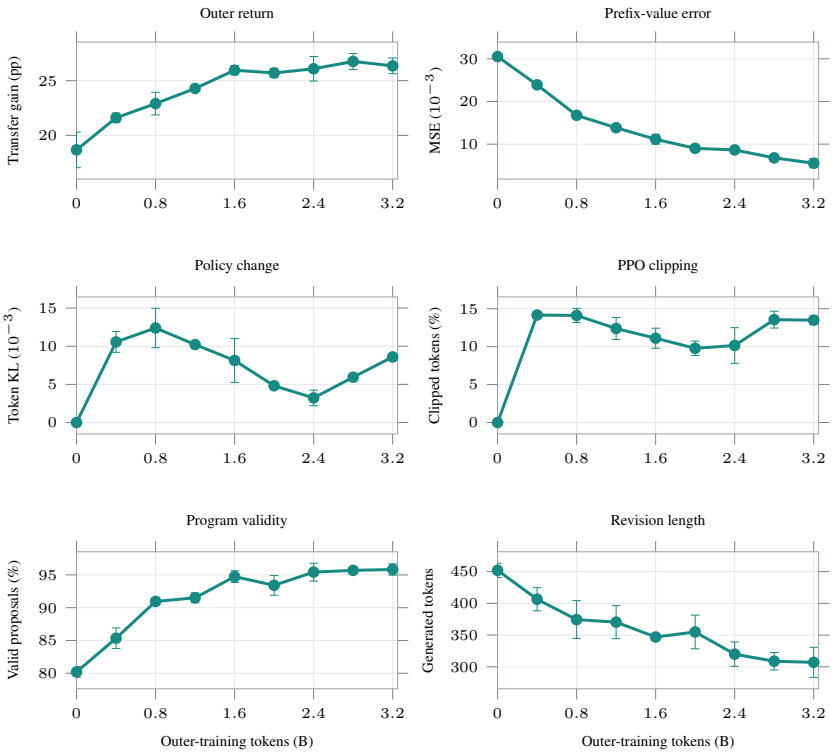

Figure 14: Feedback-Agent diagnostics across eight update rounds, each adding four complete four-stage trajectories and 0.4B inner-training tokens.

The lowest-scoring ALFWorld example loses development transfer after a valid program discards unresolved observation history. The paired Keep continuation from that same state performs better, so a positive endpoint over the initial model would not erase the local regression.

Table 9: End-to-end cost at 100M retained training tokens with shared teacher-construction cost amortized over 20 deployments. GPU-hours sum all phases and report SD across blocks.

Method	GPU-hours	Tokens (M)	Calls (k)	Tools (k)
Selected fixed protocol	45.9 ± 1.2	148.00	52.6	47.2
One-shot optimization	46.0 ± 0.2	138.00	49.4	44.3
Frozen program generator	40.9 ± 1.4	118.02	43.0	38.5
Execution-based search	44.0 ± 1.2	126.10	47.0	42.1
Transfer-based search	147.8 ± 3.0	436.03	150.0	135.0
TAME	106.5 ± 4.6	326.06	111.9	100.8

F SUPPLEMENTARY EVALUATIONS

This section presents the complete model–domain comparisons underlying the main results, including performance under each evaluation harness and variation across repeated runs. Success rates are expressed as percentages and score differences as percentage points. In all ranking plots, rank 1 denotes the highest score.

The primary common-transfer score assigns weight $1/2$ to $H_{\min}$ and $1/8$ to each of H_{docs}, H_{info}, H_{exec}, and H_{null}. Results for the diagnostic planning and retry harnesses and the held-out harness are reported separately. Attached-system performance is also kept distinct: it evaluates each model with its own final program, whereas fixed-harness scores compare models under shared deployment conditions.

Shaded bands and error bars show one standard deviation at the replication level specified in each caption. The crossed-harness study uses five paired task-model replicates.

Table 10: Exact paired primary comparisons for Qwen3-4B at 100M training tokens. Entries are mean $\pm s_{\text{fit}}$ in percentage points. Within-fit dispersion is reported separately in the records and Figure 19.

Domain	Comparison	Common	Minimal
ALFWorld	TAME minus fixed	$+5.5 \pm 0.9$	$+5.4 \pm 0.9$
ALFWorld	TAME minus frozen	$+3.2 \pm 2.1$	$+3.6 \pm 2.1$
ALFWorld	Common minus native reward	$+2.9 \pm 0.4$	$+3.9 \pm 0.4$
ScienceWorld	TAME minus fixed	$+4.4 \pm 0.5$	$+4.4 \pm 0.5$
ScienceWorld	TAME minus frozen	$+3.5 \pm 1.2$	$+3.9 \pm 1.2$
ScienceWorld	Common minus native reward	$+2.8 \pm 0.3$	$+3.8 \pm 0.2$
Telecom	TAME minus fixed	-1.7 ± 0.7	-1.1 ± 0.7
Telecom	TAME minus frozen	-1.9 ± 1.1	-1.4 ± 1.1
Telecom	Common minus native reward	$+2.8 \pm 0.7$	$+3.9 \pm 0.8$

F.1 TRAINING-BUDGET SENSITIVITY

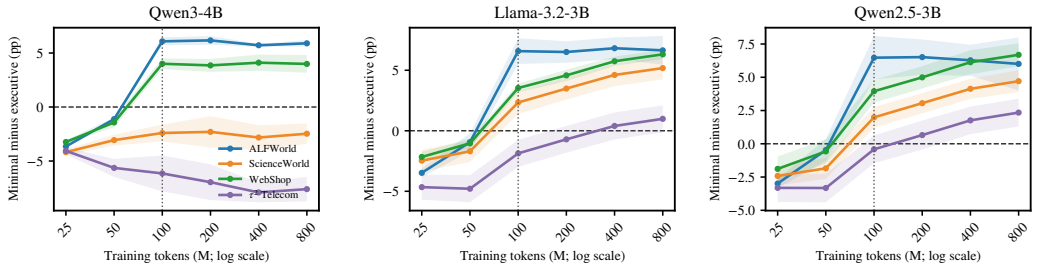

Figure 15: How rank reversal changes with training budget. Each curve is minimal-trained minus executive-trained success under the same minimal evaluation harness. Positive values favor minimal training. The dotted line marks the 100M primary comparison; bands show the standard deviation across five paired values.

F.2 Full crossed-harness results

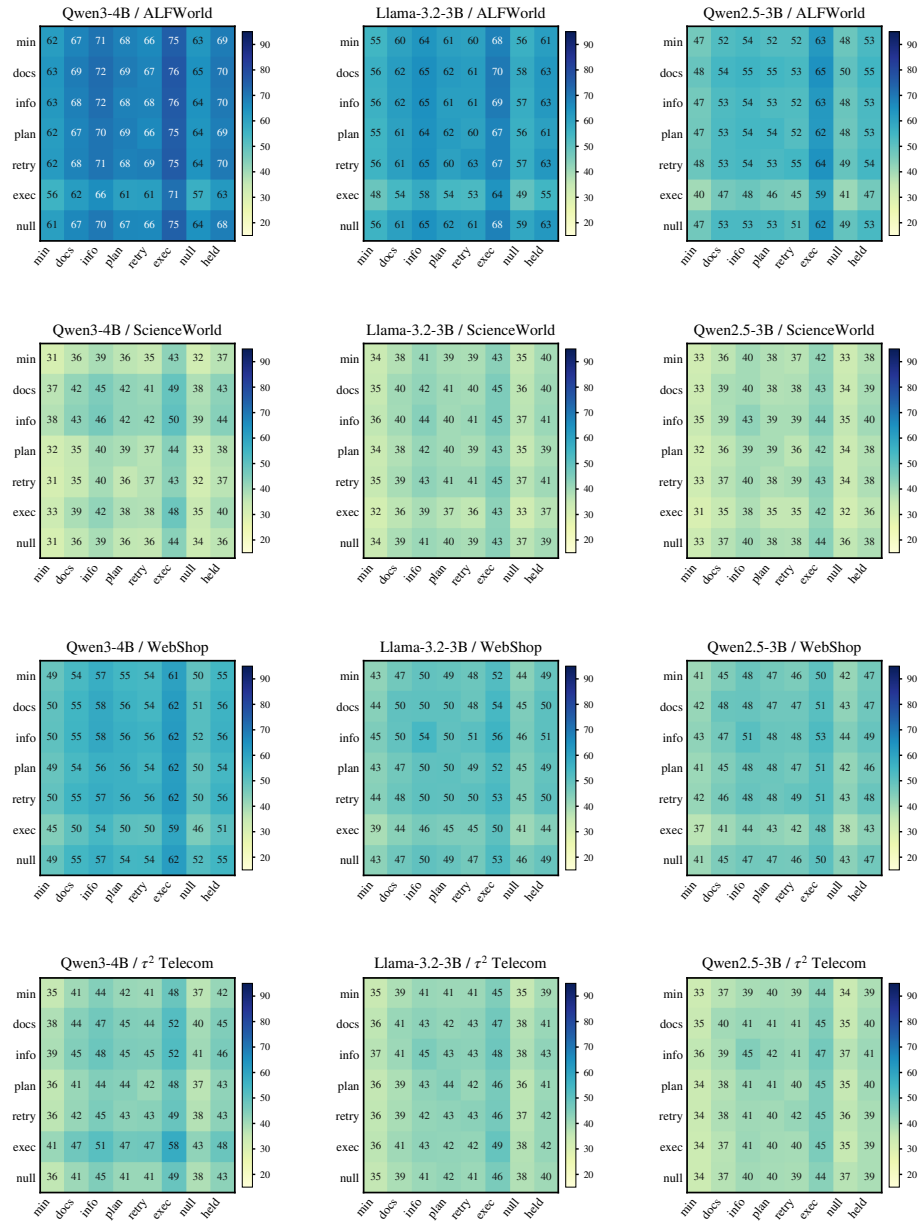

Figure 16: Crossed training and evaluation at 100M tokens. Rows specify the training harness and columns the evaluation harness; all seven training choices and eight evaluation harnesses are shown. Cells are mean success rates (%) over five paired values. The fixed 15–95 color scale is shared across panels. The corresponding records contain every matrix at 25M, 50M, 200M, 400M, and 800M tokens. Panel columns identify Qwen3-4B, Llama-3.2-3B, and Qwen2.5-3B; rows identify ALFWorld, ScienceWorld, WebShop, and Telecom.

F.3 From execution ranking to teaching ranking

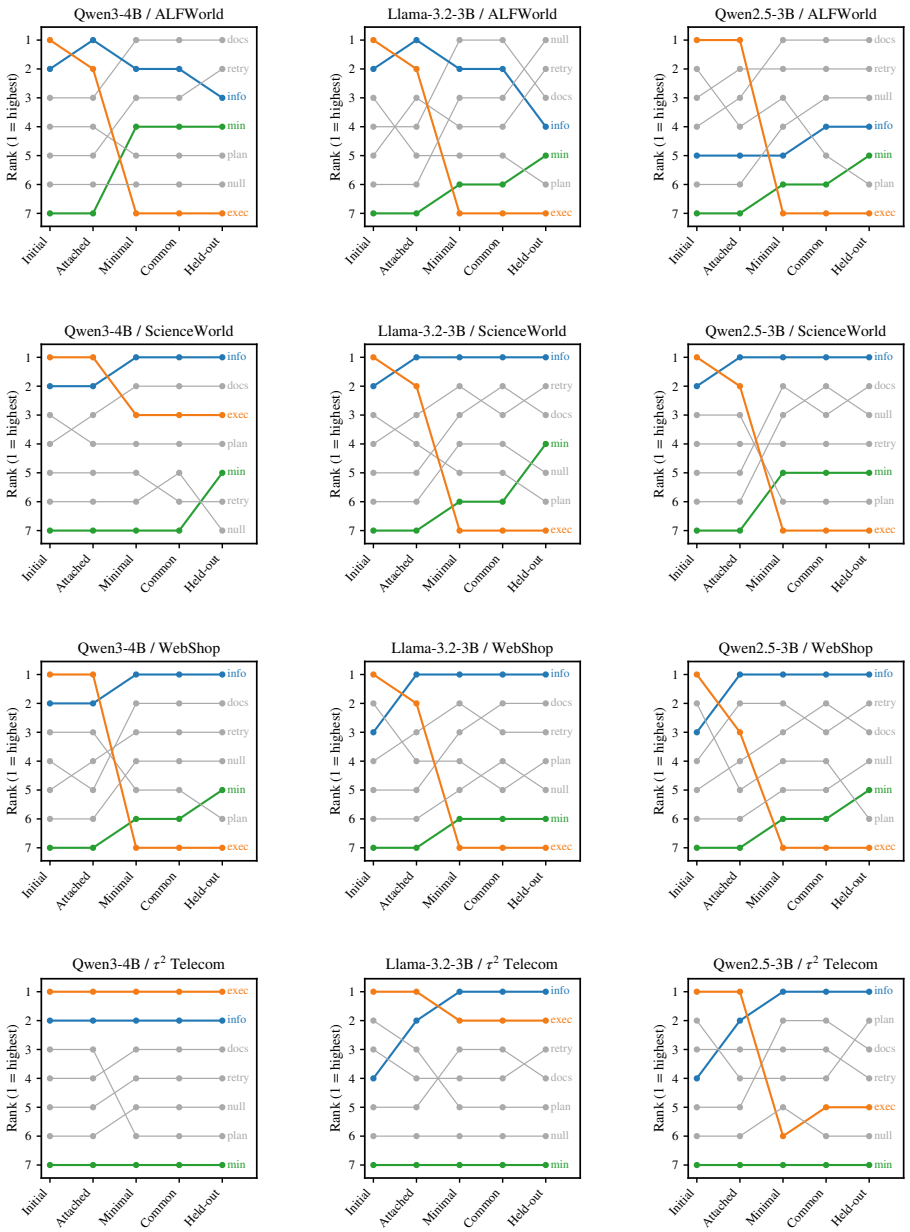

Figure 17: Five rankings of the same seven training harnesses. Initial ranks use the initial model with each attached harness. Attached ranks retain the training harness after learning; the remaining ranks use the common minimal harness, primary weighted family, or held-out harness. Rank 1 is highest. Green, blue, and orange emphasize minimal, informational, and executive training, respectively; the other harnesses are gray.

F.4 ENDPOINT PERFORMANCE ACROSS BUDGETS

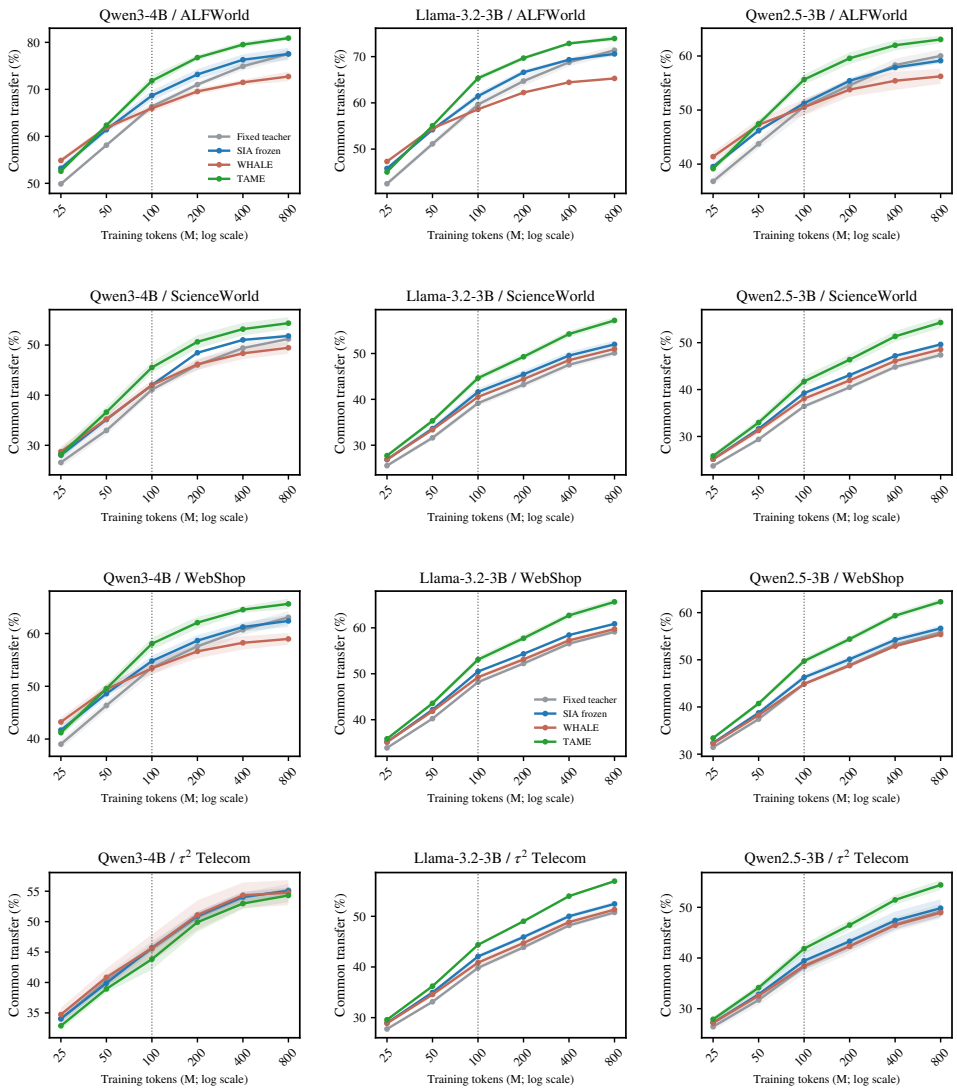

Figure 18: Common-reference performance across training budgets. All curves use the same five reference harnesses and fixed weights. After 100M tokens, the learned Feedback-Agent continues to revise the training harness. Baseline labels refer to the adaptations described in Section 6.

F.5 Paired replication and variability

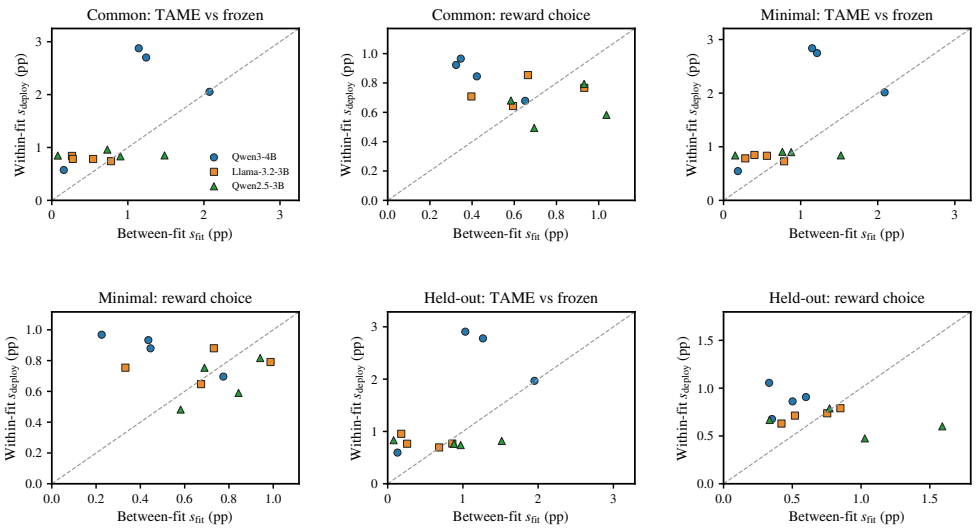

Figure 19: Two levels of dispersion for the paired comparisons. Each point is one model-domain setting. The dashed diagonal marks equal values. Columns and titles identify the performance measure and comparison.

F.6 Design ablations across models and domains

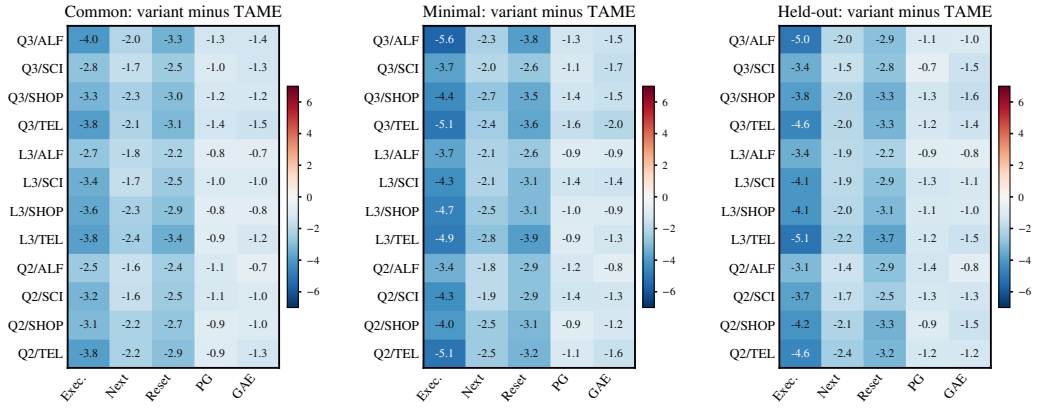

Figure 20: Design choices across all twelve settings. Columns replace transfer reward with immediate execution (Exec.), retain only next-stage credit (Next), reset persistent source (Reset), use a one-update policy gradient (PG), or set GAE to $\lambda = 0.95$. Cells show variant minus TAME in percentage points. Common, minimal, and held-out performance are kept separate.

F.7 PROGRAM REVISION AND LOCAL LEARNING EFFECTS

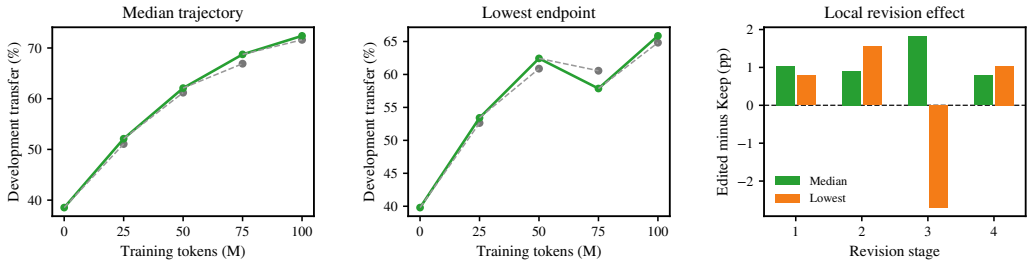

Figure 21: Revision effects on ALFWorld. The first two panels show the median and lowest final development score among the recorded Qwen3 trajectories. Gray endpoints are separate Keep continuations from each parent. The final panel subtracts each Keep endpoint from its revised endpoint. Note that a valid source edit can still have a negative teaching effect.

F.8 RESOURCE USE AND REPEATED DEPLOYMENT

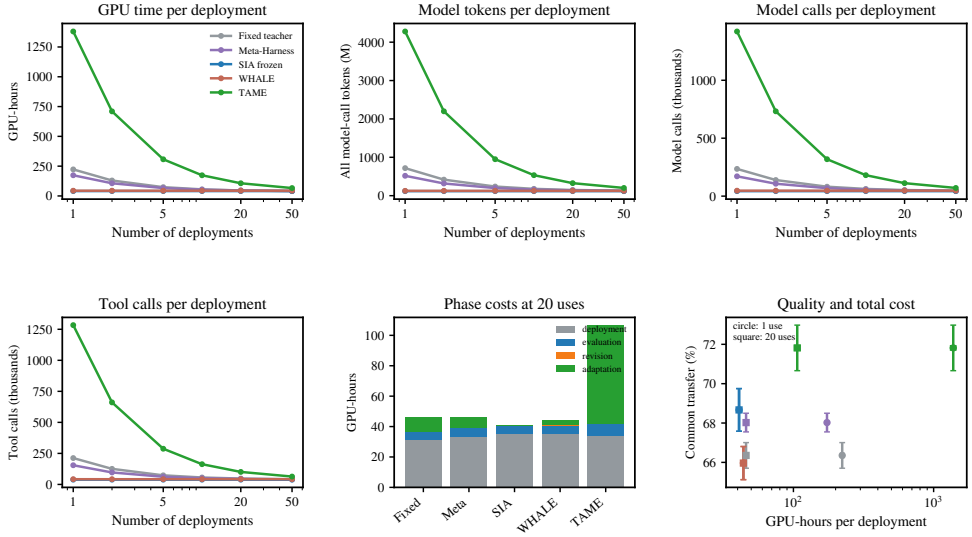

Figure 22: Resource use for the five methods with complete ALFWorld cost records at 100M task-model tokens.