CO-HARNESS: CO-EVOLVING HARNESSES AND MODEL WEIGHTS FOR TOOL-USING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training tool-using agents involves both learning model parameters and designing the runtime harness that generates their interaction trajectories. When the harness remains fixed, model updates can change agent behavior without corresponding adjustments to prompts, tools, skills, middleware, and memory. We introduce Co-Harness, a framework that alternates harness optimization with model post-training. An LLM-based HarnessCritic analyzes failed trajectories and proposes local harness edits, which are retained after validation. The model is then fine-tuned on quality-filtered trajectories collected under the updated harness, and the resulting model guides the next round of harness search. We evaluate Co-Harness on repository-level coding benchmarks SWE-bench Verified and SWE-bench Pro, and on tool-integrated mathematical reasoning benchmarks AIME24, AIME25, and HMMT25. Performance improves over two joint rounds across the evaluated model–benchmark pairs. A separate 200+ hour case study with model weights fixed documents automated runtime repair, execution-efficiency changes, and ensemble exploration. Co-Harness incorporates adaptation of the trajectory-generating runtime into the model-training process.

1 INTRODUCTION

Post-training an agent is not only about training model weights, but also about designing the runtime system around the model. In practice, an agent consists of two coupled components: model parameters and the *Harness* that elicits, executes, verifies, and records its behavior. The Harness includes prompts, tools, reusable skills, middleware, retry logic, context management, and memory. Current post-training pipelines treat these components asymmetrically. The model is updated through supervised fine-tuning, reinforcement learning, or preference optimization, while the Harness is usually hand-designed and fixed. This creates a mismatch: the Harness determines the trajectories used for training, yet it is not improved from failures observed during training. A bad tool schema, missing retry hook, short turn limit, or underspecified prompt can prevent useful trajectories from being generated. As models become more capable in software engineering and tool-use settings, the fixed Harness becomes an important bottleneck for automating AI R&D (Amodei, 2024).

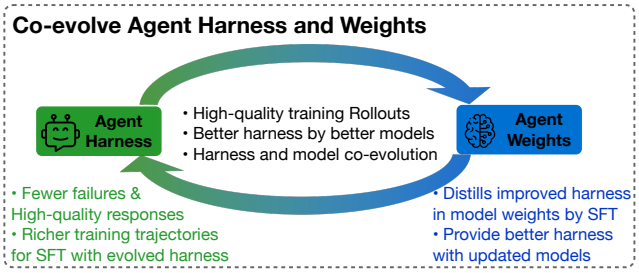

Figure 1: **Co-evolution of the harness and model.** Harness improvements enable useful training trajectories; model updates change the behavior and failures that guide subsequent harness optimization.

Existing work addresses both sides of this problem, often separately. Agent post-training learns from interaction trajectories, while harness optimization revises the system around the model using execution feedback. Recent examples include Meta-Harness, Agentic Harness Engineering, and AEvo (Lee et al., 2026; Lin et al., 2026; Zhang et al., 2026). Their editable runtime structures motivate treating the harness as an optimization variable. However, the optimized Harness is still wrapped around a fixed model. It improves how the current model runs at inference time, but it is

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.

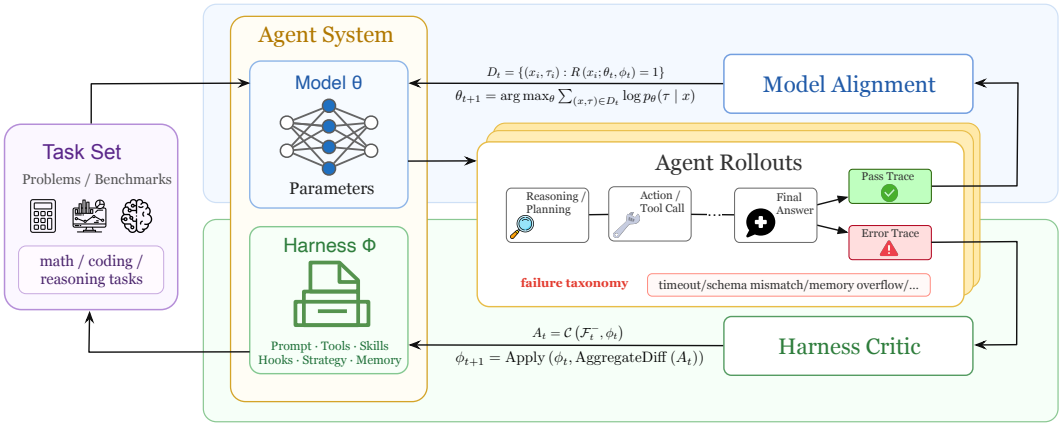

Figure 2: **Information flow through the two update paths.** Failed trajectories inform Harness-Critic's local harness edits; verified successful trajectories supply the SFT model-alignment step. The diagram summarizes the components and feedback paths.

not part of the post-training loop that produces trajectories, filters them, and updates the next model. This motivates our question: *can harness evolution be coupled with model post-training, so that the system that generates training data improves together with the model trained on that data?*

We propose **Co-Harness**[1], a simple dual-loop recipe for agent post-training. As shown in Figure 2, each round starts from a model $\boldsymbol{\theta}_t$ and a Harness $\boldsymbol{\phi}_t$. The agent first collects rollouts on a task set. Failed trajectories are passed to an LLM-based **HarnessCritic**, which attributes each failure to a structured Harness-level cause, such as prompt ambiguity, tool schema error, missing skill, or middleware mismatch. HarnessCritic then proposes local harness diffs, and a diff is accepted only if validation rollouts improve the targeted failure mode without lowering measured reward on regression-check tasks. The accepted harness $\boldsymbol{\phi}^*$ is used to collect high-quality trajectories, which are then used to fine-tune the next model $\boldsymbol{\theta}_{t+1}$. The updated model starts the next round, where it may expose new harness-level bottlenecks. The key idea is that the harness should not be treated as a fixed data-generation environment. The intended feedback is that runtime repair improves the trajectories available for learning, while model updates change the failures that guide subsequent repair:

$$\textit{Better harness} \rightarrow \textit{better trajectories} \rightarrow \textit{stronger model} \rightarrow \textit{better harness} \rightarrow \ldots$$

This makes the harness a learnable part of the post-training system, rather than a static scaffold around the model. Our contributions are:

- We formulate agent post-training as a joint optimization problem over the model and its harness. Co-Harness alternates between improving the harness from failures and training the model on trajectories generated by the improved harness.
- We introduce HarnessCritic, a failure-driven procedure for harness. Given failed trajectories, HarnessCritic diagnoses harness-level failures and proposes edits across prompts, tools, skills, middleware, and memory policies. Validation determines which edits are retained.
- We report performance gains over two joint rounds in repository-level coding and tool-integrated reasoning. A separate 200+ hour run with model weights fixed documents automated runtime repair, execution-efficiency changes, and ensemble exploration.

2 Related Work

Agent post-training. AgentTuning learns from interaction trajectories, while SWE-smith and R2E-Gym provide tasks and executable environments for training software agents (Zeng et al., 2024;

[1]Code will be released at https://github.com/AnonymousCode10/Co-Harness.

Yang et al., 2025b; Jain et al., 2025). SWE-Lego combines real and synthetic tasks with filtered trajectories for SFT (Tao et al., 2026). SWE-Master separates source tasks, rollout filtering, and SFT/RL stages (Song et al., 2026); DeepSWE uses an R2E-Gym training subset excluding repositories shared with Verified (Luo et al., 2025). STaR and ReST iterate between generating supervision and updating the model (Zelikman et al., 2022; Gulcehre et al., 2023); ReTool and ToRL study learning with executable tools (Feng et al., 2025; Li et al., 2025). Co-Harness uses quality-filtered trajectories for SFT, but also revises the runtime that produces those trajectories. It couples runtime repair with model training through the trajectories collected for SFT.

Harness optimization. DSPy, MIPRO, TextGrad, and GEPA optimize language-model programs or their textual components using feedback (Khattab et al., 2024; Opsahl-Ong et al., 2024; Yuksekgonul et al., 2025; Agrawal et al., 2025). ExpeL and ACE retain experience in reusable textual contexts (Zhao et al., 2024; Zhang et al., 2025), while ADAS searches over agent implementations (Hu et al., 2025). SWE-agent demonstrates the importance of the agent–computer interface for coding (Yang et al., 2024). More directly, Meta-Harness searches harness code using source, scores, and execution traces (Lee et al., 2026); Agentic Harness Engineering structures editable components and diagnostic evidence (Lin et al., 2026); AEvo revises the procedure or context governing agentic evolution (Zhang et al., 2026). These works motivate an editable runtime. Co-Harness connects accepted runtime edits to new training trajectories, then resumes harness search with the updated model.

3 METHOD

Co-Harness alternates harness improvement and model improvement. The working hypothesis is that runtime repair changes the supervision available to the model, while model updates change which runtime failures are useful to address next. The framework is summarized in Figure 2; Algorithm 1 gives the step-by-step execution of one joint round.

3.1 PROBLEM FORMULATION

Let π_θ be a language model and H_ϕ its runtime harness. We represent the harness as

$$\phi = (P, T, S, W, M), \tag{1}$$

where P specifies prompts, T tool interfaces, S reusable skills, W middleware, and M memory policies. Middleware includes runtime strategies, execution control, interception hooks, and context management; Hooks and Strategy in Figure 2 both belong to W. An episode on task x produces a trajectory $\tau \sim p_{\theta,\phi}(\tau \mid x)$ and an outcome $v(x, \tau)$ from a task verifier. The expected task reward is

$$J(\theta, \phi) = \mathbb{E}_{x\sim\mathcal{X},\, \tau\sim p_{\theta,\phi}(\cdot|x)}[v(x, \tau)]. \tag{2}$$

The joint objective is $\max_{\theta,\phi} J(\theta, \phi)$. We pursue this objective through alternating harness search and supervised model updates. Harness candidates require costly rollouts, outcomes are sparse, and model updates change the trajectory distribution under which a harness was selected.

A harness edit changes more than a final answer. It changes the observations the model sees, the actions it can execute, and which episodes survive long enough to produce supervision. Co-Harness therefore alternates a harness update with a model update, using task verification to connect the two.

3.2 ALTERNATING OPTIMIZATION

We distinguish initialization from joint optimization. Starting with model θ_0 and human-designed harness ϕ_{H}, an initial harness-only pass produces ϕ_0. The evaluated pair (θ_0, ϕ_0) is **R0**; no model training has yet occurred. For joint round $t + 1$, the input is (θ_t, ϕ_t):

$$\phi_t^\star = \mathrm{HarnessUpdate}(\theta_t, \phi_t), \tag{3}$$

$$\mathcal{D}_t = \{(x, \tau) : \tau \sim p_{\theta_t,\phi_t^\star}(\cdot \mid x),\ q(x, \tau) = 1\}, \tag{4}$$

$$\theta_{t+1} = \mathrm{SFT}(\theta_t; \mathcal{D}_t). \tag{5}$$

R1 evaluates (θ_1, ϕ_1) and **R2** evaluates (θ_2, ϕ_2). The harness inherited by the next round is $\phi_{t+1} \leftarrow \phi_t^\star$. Here, $q(x, \tau) = 1$ denotes passing the task verifier or the benchmark-specific quality filters;

Table 1: **Stage definitions.** Each joint round updates the harness before collecting SFT trajectories. The initial Human→R0 search is separate from the harness update inside R1.

Stage	Evaluated pair	Operations since the preceding stage
Human	$(\theta_0, \phi_{\mathrm{H}})$	Human-authored starting harness; no updates.
R0	(θ_0, ϕ_0)	Initial harness-only search; model fixed.
R1	(θ_1, ϕ_1)	Search with θ_0 to obtain ϕ_1; collect $\mathcal{D}_0$ with (θ_0, ϕ_1); SFT to obtain θ_1.
R2	(θ_2, ϕ_2)	Search with θ_1 to obtain ϕ_2; collect $\mathcal{D}_1$ with (θ_1, ϕ_2); SFT to obtain θ_2.

Algorithm 1 One joint round of Co-Harness

Require: Model θ_t, harness ϕ_t, optimization tasks, validation tasks, inner budget K

$\phi \leftarrow \phi_t$
for $k = 1, \ldots, K$ **do**
  Collect failures $\mathcal{F}^-$ under (θ_t, ϕ)
  $\mathcal{A}_t \leftarrow \mathcal{C}(\mathcal{F}^-, \phi)$; $\Delta \leftarrow \mathrm{AggregateDiffs}(\mathcal{A}_t)$
  Validate candidate $\widetilde{\phi} = \mathrm{Apply}(\phi, \Delta)$ against ϕ
  if targeted performance improves and regression checks pass **then**
    Commit candidate and set $\phi \leftarrow \widetilde{\phi}$
  else
    Reject candidate and retain ϕ
  end if
end for
Set $\phi_t^\star \leftarrow \phi$; collect quality-filtered trajectories $\mathcal{D}_t$
Train $\theta_{t+1} \leftarrow \mathrm{SFT}(\theta_t; \mathcal{D}_t)$
Set $\phi_{t+1} \leftarrow \phi_t^\star$
return $(\theta_{t+1}, \phi_{t+1})$

v remains the task outcome used for evaluation. This convention separates the gain from initial harness search from subsequent gains after model training and renewed harness search. The general schedule stops when no candidate passes validation or reward gains remain below a small threshold for consecutive rounds; the reported TIR experiments instead use a fixed two-round horizon.

3.3 Harness Update

HarnessCritic, denoted $\mathcal{C}$, receives failed trajectories and the active harness configuration. Its output is the attribution set $\mathcal{A}_t = \mathcal{C}(\mathcal{F}^-, \phi)$, containing one structured record per failure. The local ϕ is updated during the transition from t to $t+1$, starting from ϕ_t. Each record contains a root-cause label, the implicated harness component, severity, evidence from the trace, and a proposed edit. The actionable labels cover prompt ambiguity, tool-interface errors, missing skills, middleware mismatches, and memory failures. A separate model-side label permits abstention: an incorrect derivation or an incorrect patch does not automatically justify editing the harness. Sections C and G give the taxonomy, initial TIR configuration, and attribution interface.

The attribution interface separates trace evidence from the proposed intervention. For example, repeated tool calls can reflect a rejected argument schema, a middleware policy that leaves an unproductive loop running, or a model-side reasoning error. The trace evidence identifies the event to explain; the component label and field path locate a candidate change. Its old and new values specify the intervention to validate. Records labeled `agent_error` are excluded from patch generation.

HarnessCritic proposes failure diagnoses for validation. We rank candidate edits by recurrence, severity, and consistency of the proposed field path, then combine compatible suggestions with $\mathrm{AggregateDiffs}(\mathcal{A}_t)$. Each candidate records the affected field or code region and its prior value, making the intervention inspectable and reversible. Concrete candidates include correcting a tool schema, changing a retry hook, or adjusting output truncation. Recurrence and severity prioritize the search, and validation determines which candidates are retained. When several edits are combined, validation measures their joint effect at the configuration level.

The acceptance rule compares a candidate to the current harness with the model fixed. On validation subsets targeting the failure and checking other behaviors, respectively, it requires

$$\Delta J_{\text{target}} > 0, \qquad \Delta J_{\text{check}} \geq 0. \tag{6}$$

Accepted candidates enter a versioned registry for subsequent use and comparison; rejected candidates leave the current harness unchanged. Here, ΔJ denotes the candidate-minus-current empirical reward on the corresponding validation subset. The regression check is scoped to that subset; final performance is measured separately on held-out evaluation tasks.

3.4 Model Update

With $\phi_t^\star$ fixed, we re-run the current model and retain trajectories that pass the task verifier or the benchmark-specific quality filters, then apply supervised fine-tuning. A trajectory includes the interaction context, model actions, tool observations, and verified outcome. The SFT objective applies to model response tokens conditioned on the interaction context. In shorthand, the update optimizes

$$\mathcal{L}(\theta; \mathcal{D}_t) = - \sum_{(x,\tau)\in\mathcal{D}_t} \log p_{\theta,\phi_t^\star}(\tau \mid x), \tag{7}$$

where the model likelihood applies to generated decisions rather than to the external tool transition itself. The purpose is to internalize behavior enabled by the improved harness. The resulting model may expose new runtime bottlenecks, motivating the next harness update. Whether the resulting model can retain that behavior under a different harness is a separate transfer question, not guaranteed by the training objective. In particular, executable infrastructure such as process cleanup remains part of the harness; SFT can learn response and tool-use behavior, not replace that infrastructure.

4 Experiments

We evaluate Co-Harness on repository-level coding and tool-integrated reasoning (TIR). The main results report the final scores of each model–harness pair; Section 5 examines how that capability develops across rounds, model scales, and harness revisions.

4.1 Experimental Setup

Models and stages. Both task families use Qwen3-8B and Qwen3-32B (Yang et al., 2025a). **Human** is the initial model with a static human-designed harness. **R0** follows an initial harness-only search with model weights fixed. **R1** and **R2** follow one and two joint rounds, each updating the harness and then fine-tuning the model on successful trajectories (Section 3.2). Human→R0 therefore measures the initial harness update, whereas R0→R2 measures the subsequent joint schedule. Table 1 specifies the evaluated pairs; full stage-wise results are in Section B. Human supplies the starting configuration for harness search. The initial TIR configuration is given in Section G.

Coding tasks and metric. SWE-bench Verified and SWE-bench Pro evaluate repository-level issue resolution through executable tests (Jimenez et al., 2024; Chowdhury et al., 2024; Deng et al., 2025). Agents navigate repositories, edit code, and execute commands; patch verification identifies successful training trajectories. We report resolved-task percentages. Pro uses the same 731-task public dataset version as the published references. Repository tools are editable interfaces; benchmark definitions and scoring logic are outside the edit surface.

TIR tasks and metric. AIME24, AIME25, and HMMT25 each contain 30 problems. Agents interleave reasoning with Python execution, and a verifier checks the final answer. The harness controls call parsing, interpreter lifecycle, turn limits, and returned observations. Each joint round uses $K = 5$ inner harness-search iterations followed by SFT. We estimate pass@1 by averaging 32 rollouts per problem. TIR summary averages weight the six model–benchmark pairs equally. Coding results are reported separately. The extended case study uses a separate six-trajectory majority-vote ensemble. Section A details reporting conventions. The starting configuration in Section G uses mathematical-reasoning and Python-use instructions with no few-shot retrieval. Calls are dispatched to a sandboxed Python subprocess with a 30-second timeout. Middleware permits up to three retries

Table 2: **Coding results by model scale.** Resolved tasks (%). †: published single-attempt results; unmarked: our aggregate evaluations. Bold identifies Co-Harness; underlining marks the highest reported score within each benchmark's 8B/32B group. Harnesses and budgets differ across systems. Full round-wise results: Table 4.

Model / Method	Harness	Resolved
(a) SWE-bench Verified		
Closed-source references		
Claude 3.7 Sonnet$^{\dagger S}$	SWE-agent	58.2
Larger open-weight references		
Qwen3-235B-A22B$^{\dagger S}$	OpenHands	34.4
Open-weight dense models: 8B		
SWE-Lego-Qwen3-8B$^{\dagger L}$	OpenHands	42.2
Qwen3-8B (Human)	Human-designed	5.0
Co-Harness-8B (Ours)	Co-evolved	**43.0**
Open-weight dense models: 32B		
SWE-Lego-Qwen3-32B$^{\dagger L}$	OpenHands	52.6
DeepSWE-Preview-32B$^{\dagger E}$	R2E-Gym	42.2
SWE-Master-32B-RL$^{\dagger M}$	R2E-Gym	61.4
Qwen3-32B (Human)	Human-designed	12.9
Co-Harness-32B (Ours)	Co-evolved	**61.8**
(b) SWE-bench Pro		
Public split: 731 tasks		
Closed-source references		
GPT-5 (high)$^{\dagger P}$	SWE-agent	41.8
Claude Sonnet 4.5$^{\dagger P}$	SWE-agent	43.6
Larger open-weight references		
GPT-OSS-120B$^{\dagger P}$	SWE-agent	16.2
Kimi K2 Instruct$^{\dagger P}$	SWE-agent	27.7
Open-weight dense models: 8B		
Qwen3-8B (Human)	Human-designed	0.2
Co-Harness-8B (Ours)	Co-evolved	**18.1**
Open-weight dense models: 32B		
Qwen3-32B (Human)	Human-designed	3.4
Co-Harness-32B (Ours)	Co-evolved	**30.6**

S: Yang et al. (2025b); L: Tao et al. (2026); E: Luo et al. (2025); M: Song et al. (2026); P: Deng et al. (2025). Our rows use R2. S: historical scores compiled in SWE-smith. L: Git-hacking-prevented scores. P: Pro paper v2, not the later 642-task dataset V2. No multi-candidate TTS scores are included. Protocols: Section A.1.

on tool-call exceptions with linear backoff, initially limits episodes to 15 turns, and stops on a completion marker or final-answer token. Context compression retains the last 32k tokens, while interpreter outputs are truncated to 2k tokens. Reusable skills and persistent memory are initially absent. The extended case later increases the turn limit to 30 (Section 5.3).

Reporting scope. Results are point estimates for the evaluated model–harness configurations. TIR pass@1 averages repeated single-attempt outcomes from each configuration. Uncertainty reporting and published baseline protocols are detailed in Sections A and A.1.

4.2 Main Results across Coding and TIR

Coding. Table 2 reports final Verified scores of 43.0% and 61.8%, and Pro scores of 18.1% and 30.6%, for 8B and 32B. On Verified, the 32B score is numerically higher than the listed 32B references, exceeding SWE-Lego by 9.2 pp and SWE-Master by 0.4 pp. SWE-Lego and DeepSWE provide same-backbone references; historical larger-model systems contextualize the absolute scores. Pro references range from GPT-OSS-120B and Kimi K2 to GPT-5 and Claude Sonnet 4.5.

TIR. In Table 3, the 8B scores of 86.7%, 78.3%, and 62.9% are the highest among the listed 8B models, exceeding DeepSeek's Qwen3-8B distillation by 0.7, 2.0, and 1.4 pp. The 32B scores exceed the stronger ReTool-R1-32B reference by 14.8 pp on AIME24 and 32.0 pp on AIME25; ReTool

Table 3: **TIR results by model scale.** Accuracy (%). †: published results; unmarked: our evaluations. Bold identifies Co-Harness; underlining marks the highest reported score per task among the listed models in each dense 8B or 32B group, not a matched-protocol ranking. Tools denotes code-interpreter access. Full round-wise results: Table 5.

Model / Method	Tools	AIME24	AIME25	HMMT25
Closed-source references				
Gemini-2.5-Flash-Thinking-0520$^{\dagger D}$	No	82.3	72.0	64.2
o3-mini (medium)$^{\dagger D}$	No	79.6	76.7	53.3
Other open-weight scales / MoE				
DeepSeek-R1-0528 (671B)$^{\dagger D}$	No	91.4	87.5	79.4
Qwen3-235B-A22B (Thinking)$^{\dagger D}$	No	85.7	81.5	62.5
Open-weight dense models: 8B				
DeepSeek-R1-0528-Qwen3-8B$^{\dagger D}$	No	86.0	76.3	61.5
Qwen3-8B (Human)	Yes	59.3	51.3	34.7
Co-Harness-8B (Ours)	Yes	**86.7**	**78.3**	**62.9**
Open-weight dense models: 32B				
ReTool-Qwen-32B$^{\dagger R}$	Yes	67.0	49.3	—
ReTool-R1-32B$^{\dagger R}$	Yes	72.5	54.3	—
Qwen3-32B (Human)	Yes	72.0	61.3	46.7
Co-Harness-32B (Ours)	Yes	**87.3**	**86.3**	**77.0**

D: DeepSeek-AI (2025); R: Feng et al. (2025). Our rows use R2. D includes transcribed closed-model references. ReTool-Qwen/R1 use Qwen2.5-Instruct/DeepSeek-R1-Distill backbones. Qwen3-235B is the original Thinking model; 235B/22B denotes MoE total/active size. Protocols and missing entries: Section A.1.

uses a different backbone and does not report HMMT25. The 32B results also exceed the original Qwen3-235B-A22B Thinking reference on all three tasks, while full-size DeepSeek-R1-0528 scores higher on each benchmark.

The reference results use the harnesses and inference budgets specified in their source publications (Section A.1). Section 5 examines the stage-wise improvements within Co-Harness.

5 Analysis and Discussion

5.1 Gains across Optimization Rounds

Figure 3 shows the complete progression for both task families. All ten model–benchmark pairs improve after the initial harness-only update and after each joint round. On Coding, R0→R2 gains are 34.6 and 42.6 pp on Verified and 17.3 and 25.8 pp on Pro for 8B and 32B, respectively. Mean TIR accuracy rises from 58.5% at R0 to 73.5% at R1 and 79.8% at R2, a cumulative gain of 21.2 pp over R0 and 25.5 pp over Human.

The second joint round adds 11.4 and 19.6 pp on Verified, compared with 3.8 and 3.0 pp on Pro. Across TIR pairs, it adds 6.3 pp on average, accounting for 29.5% of the joint-round gain. Nine of ten pairs have a smaller second-round increment than first-round increment; the exception is 8B AIME25 (11.6 versus 9.8 pp). Thus, the reported scores increase after another round in both domains, but the increment varies substantially (Table 6).

5.2 Effects of Model Scale and Task

The 32B model achieves higher final scores on all five benchmarks, but larger size does not uniformly increase the benefit of an update. Its second-round Verified gain exceeds that of 8B (19.6 versus 11.4 pp), whereas its Pro gain is smaller (3.0 versus 3.8 pp). TIR likewise shows different scale effects: R0→R2 gains on AIME24 favor 8B (23.4 versus 10.6 pp), while HMMT25 favors 32B (27.2 versus 23.3 pp); AIME25 gains are equal at 21.4 pp. The smaller model therefore has the larger average TIR improvement (22.7 versus 19.7 pp), despite lower final accuracy. This task dependence argues against assigning a single saturation point to a model size.

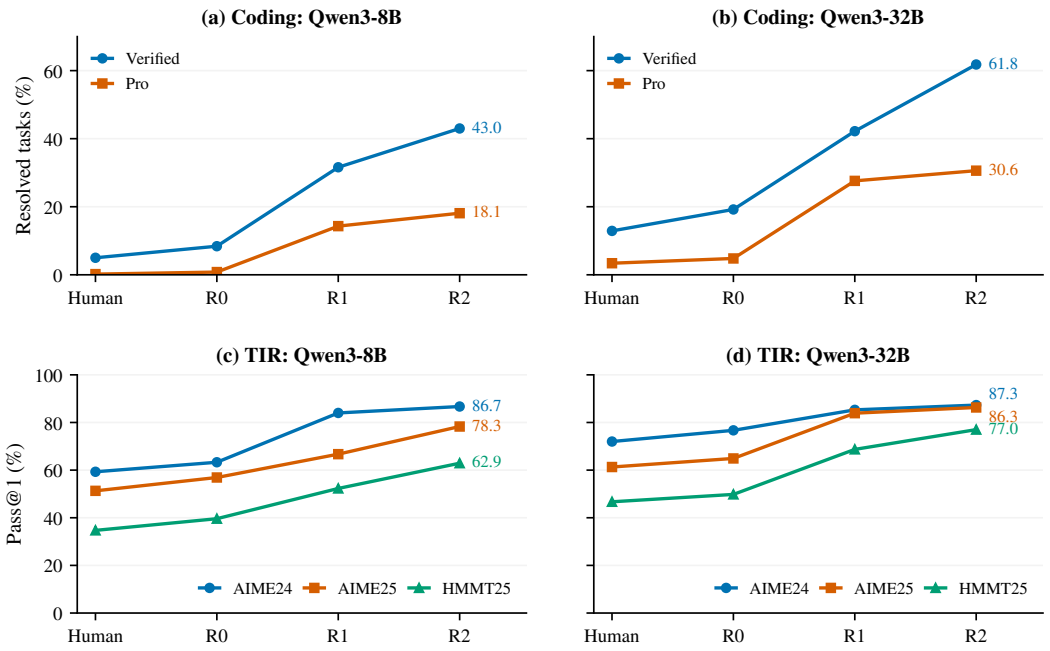

Figure 3: **Continued gains across both task families.** Columns show model sizes; rows show Coding and TIR with their respective metrics. Human→R0 updates only the harness; R1 and R2 each add a harness update and SFT. Endpoint labels report R2 scores.

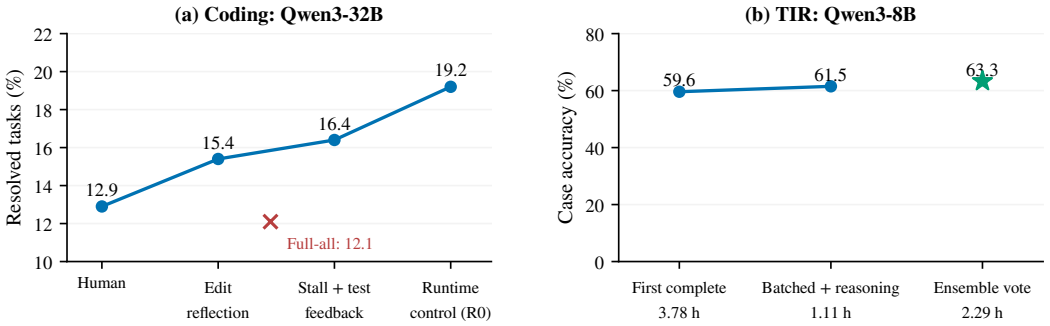

Figure 4: **Selected harness-search outcomes in Coding and TIR.** Left: selected milestones in the 32B Verified search; full-all is a regressing alternative, and 19.2% is R0, not R2. Right: complete 8B AIME24 configurations, with runtimes below the labels; the star denotes six-trajectory voting, whereas the preceding points are single configurations. Sections E and F provide detailed case records and supplementary figures.

5.3 What Changes during Harness Evolution?

Figure 4 summarizes two searches with model weights fixed. These cases examine concrete runtime changes; they are distinct from the joint-round progression above.

Coding: selective intervention rather than stacking. Across 28 Verified configurations, edit reflection raises the 12.9% baseline to 15.4%. Combining stall handling and test-error analysis reaches 16.4%; adding submission gating, scope checks, and repetitive-probe detection reaches 19.2%, a 6.3 pp gain before model training. In contrast, stacking the initial interventions yields 12.1%, below the baseline. Later prompt variants also regress. The search illustrates why plausible edits need evaluation rather than unconditional accumulation (Figure 5 and Table 10). In the selected configuration, repetitive-probe detection uses a sequence-similarity threshold of 0.85 and issues a soft

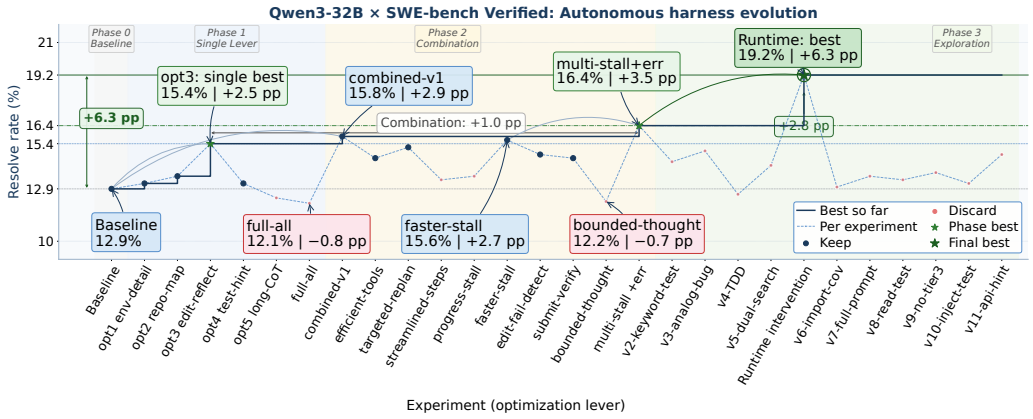

Figure 5: **Qwen3-32B harness evolution on SWE-bench Verified.** All 28 configurations are shown; the staircase tracks the best score. The selected harness improves Human (12.9%) to R0 (19.2%) with model weights fixed. Box gains are relative to the 12.9% baseline. Configuration names and scores are listed in Table 10.

hint after three similar probes. This intervention changes the feedback delivered during execution; its contribution is not separated from the accompanying submission and scope checks. Supplementary results with LongCat models (Meituan LongCat Team et al., 2025) provide additional fixed-model Coding evidence (Section D).

TIR: recovery, execution efficiency, and rollback. The 200+ hour, 22-version AIME24 case first repairs an inference-backend configuration error and subprocess cleanup. The first completed run scores 59.6% in 3.78 hours; a later configuration combining batching, follow-up reasoning, and a larger turn limit scores 61.5% in 1.11 hours. The 3.4× runtime ratio compares complete configurations, not batching alone. A six-trajectory ensemble reaches 63.3% in 2.29 hours, within the 3.33-hour budget; subsequent regressing prompt edits are rolled back. This ensemble uses a different aggregation rule from main-table pass@1 (Figure 6 and Table 11). The intermediate v12 configuration scores 50.5% after batching is introduced; follow-up reasoning is then restored. The earlier v5 score of 63.3% exceeds the time budget, so accuracy and feasibility must be read together.

Scope of harness evolution. Both searches operate within supplied task environments and modify the runtime while holding model weights fixed. Their revision histories show how candidate edits, measured regressions, and rollback shape the selected configuration.

6 Conclusion

Co-Harness alternates failure-driven harness repair with supervised learning from trajectories collected under the updated runtime. Across repository-level coding and mathematical TIR, the evaluated Qwen3 model–harness pairs improve over both joint rounds, with gains that vary by task and model size. Fixed-model cases document runtime recovery, execution-efficiency changes, and rollback after regressions. Co-Harness incorporates runtime adaptation into post-training, allowing each model update to learn from trajectories generated by the revised harness.

Limitations. Our evaluation covers a limited set of models and benchmarks, and the findings may not generalize to other model families, task domains, or deployment conditions. The method also requires repeated trajectory generation and validation, which can incur substantial computational cost. Performance depends on the quality of the underlying models, training data, and evaluation feedback. HarnessCritic depends on a capable model and repeated validation rollouts. Broader evaluation is needed to characterize robustness, efficiency, and reliability in practical applications.

AI USE STATEMENT

Generative AI tools assisted with manuscript editing, LaTeX and figure/table code, literature discovery, feedback on methodology presentation, and structured review. They did not generate the reported experimental measurements.

ETHICS STATEMENT

Co-Harness adapts the runtime of tool-using agents, including code execution and control policies. Such changes can introduce unintended behavior, expose sensitive information through execution traces, or weaken safeguards even when benchmark performance improves. Evaluation gains therefore should not be interpreted as evidence of safe deployment. Deployment should enforce execution isolation, access controls, and resource limits outside the editable harness, with human review of changes affecting security boundaries. Trajectories used for diagnosis or training should be checked for credentials, personal information, and restrictions on data reuse. Our evaluation is limited to the reported benchmark settings and does not establish safety in production environments.

REPRODUCIBILITY STATEMENT

The optimization variables, round indexing, and acceptance rule are specified in Section 3; benchmark stages, metrics, and aggregation appear in Section 4. Section A describes the evaluation metrics, aggregation conventions, and literature-reference protocols.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, et al. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. *arXiv preprint arXiv:2507.19457*, 2025. URL https://arxiv.org/abs/2507.19457.

Dario Amodei. Machines of loving grace. Essay, https://www.darioamodei.com/essay/machines-of-loving-grace, 2024.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, et al. Introducing SWE-bench verified. OpenAI, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

DeepSeek-AI. DeepSeek-R1-0528 and DeepSeek-R1-0528-Qwen3-8B: Model Card. Official model card, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-R1-0528-Qwen3-8B. Accessed September 25, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, et al. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, et al. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs. *arXiv preprint arXiv:2504.11536*, 2025. URL https://arxiv.org/abs/2504.11536.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, et al. Reinforced Self-Training (ReST) for Language Modeling. *arXiv preprint arXiv:2308.08998*, 2023. URL https://arxiv.org/abs/2308.08998.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.08435.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural Environments and Hybrid Verifiers for Scaling Open-Weights SWE Agents. *arXiv preprint arXiv:2504.07164*, 2025. URL https://arxiv.org/abs/2504.07164.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

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.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling Tool-Integrated RL. *arXiv preprint arXiv:2503.23383*, 2025. URL https://arxiv.org/abs/2503.23383.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, et al. Agentic Harness Engineering: Observability-Driven Automatic Evolution of Coding-Agent Harnesses. *arXiv preprint arXiv:2604.25850*, 2026. URL https://arxiv.org/abs/2604.25850.

Hong Liu, Jiaqi Zhang, Chao Wang, Xing Hu, Linkun Lyu, Jiaqi Sun, Xurui Yang, Bo Wang, Fengcun Li, Yulei Qian, Lingtong Si, Yerui Sun, Rumei Li, Peng Pei, Yuchen Xie, and Xunliang Cai. Scaling Embeddings Outperforms Scaling Experts in Language Models. *arXiv preprint arXiv:2601.21204*, 2026. URL https://arxiv.org/abs/2601.21204.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE-Preview: Model Card and Evaluation. Official model card, 2025. URL https://huggingface.co/agentica-org/DeepSWE-Preview. Accessed September 25, 2026.

Meituan LongCat Team et al. LongCat-Flash Technical Report. *arXiv preprint arXiv:2509.01322*, 2025. URL https://arxiv.org/abs/2509.01322.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Scale AI. SWE-bench Pro V2: Dataset and Evaluation Protocol. Official dataset repository, 2026. URL https://github.com/scaleapi/SWE-bench_Pro-os/blob/main/v2/README.md. Accessed September 25, 2026.

Huatong Song, Lisheng Huang, Shuang Sun, Jinhao Jiang, Ran Le, Daixuan Cheng, Guoxin Chen, Yiwen Hu, Zongchao Chen, Yiming Jia, Wayne Xin Zhao, Yang Song, Tao Zhang, and Ji-Rong Wen. SWE-Master: Unleashing the Potential of Software Engineering Agents via Post-Training. *arXiv preprint arXiv:2602.03411*, 2026. URL https://arxiv.org/abs/2602.03411v1.

Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, Zhiming Mao, Xinyu Wang, Lifeng Shang, and Haoli Bai. SWE-Lego: Pushing the Limits of Supervised Fine-tuning for Software Issue Resolving. *arXiv preprint arXiv:2601.01426*, 2026. URL https://arxiv.org/abs/2601.01426v2.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

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

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, et al. SWE-smith: Scaling Data for Software Engineering Agents. *arXiv preprint arXiv:2504.21798*, 2025b. URL https://arxiv.org/abs/2504.21798.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, et al. Harnessing Agentic Evolution. *arXiv preprint arXiv:2605.13821*, 2026. URL https://arxiv.org/abs/2605.13821.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

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

A Experimental Protocols and Reporting

We evaluate Qwen models on coding and TIR tasks and report supplementary LongCat experiments separately. Averages use available scores; missing entries are omitted.

Coding evaluation. We evaluate Qwen3-8B and Qwen3-32B on SWE-bench Verified and the 731-task public split of SWE-bench Pro. Pro uses the same dataset version as the published references in Table 2. We report resolved-task percentages at Human, R0, R1, and R2; Table 4 contains all 16 model–benchmark–stage scores.

TIR evaluation. We evaluate Qwen3-8B and Qwen3-32B on AIME24, AIME25, and HMMT25, with 30 problems per benchmark. Each of the two joint rounds uses five inner harness-search iterations followed by SFT. Main-table scores are pass@1 estimates obtained by averaging 32 rollouts per problem. The extended harness-evolution case uses six-trajectory majority voting for its ensemble.

Coding SFT hyperparameters. We follow the SWE-Lego SFT hyperparameter recipe (Tao et al., 2026) for the Coding model updates. Each SFT stage uses full-parameter fine-tuning for four epochs with a global batch size of 64. The learning rate is 1×10^{-4} for Qwen3-8B and 5×10^{-5} for Qwen3-32B. We use AdamW with weight decay 0.01, a cosine learning-rate schedule, and a warmup ratio of 0.1. These optimization settings apply to the trajectory-based model update in each joint round; the round ordering and trajectory collection procedure follow Section 3.2. The full-parameter updates apply to the Qwen3 task models.

TIR decoding. Following the ReTool evaluation settings (Feng et al., 2025), we sample at temperature 1.0 and top-p 0.7. We estimate pass@1 as the mean correctness of the 32 sampled responses per problem. The extended case study reports six-trajectory majority-vote accuracy separately.

Data roles and reproducibility. Training trajectories supply supervision for SFT; failed trajectories guide harness edits; validation outcomes determine which edits are retained; final evaluation measures the resulting model–harness pair. Final evaluation tasks are held out at the instance level from training-trajectory collection, failure diagnosis, and harness-selection validation. Neither trajectories nor verifier feedback from the held-out evaluation instances are used for SFT or harness optimization. This separation supports evaluation of held-out generalization within the reported benchmarks.

Uncertainty. The reported scores are point estimates. The 32 rollouts per TIR problem measure repeated decoding from the same trained model–harness configuration.

Interpretation of the averages. All TIR averages are unweighted means of our six model–benchmark scores in Table 5; external reference models are excluded. Differences are calculated before rounding to one decimal place. Relative improvements, when used, divide the score difference by the stated starting score; percentage-point changes subtract scores directly. The second-round share divides the mean R2−R1 difference by the mean R2−R0 difference.

A.1 Literature Reference Provenance

The dagger-marked entries in Tables 2 and 3 are published results obtained with each source's harness and inference budget. These protocols vary across systems and are specified below. Source letters identify the publications or official model cards. Models are grouped by size, with MoE models listed separately from dense models. Boldface identifies our R2 rows; underlining indicates the highest reported value within each dense-model group.

TIR sources. D uses the official DeepSeek-R1-0528 model card (DeepSeek-AI, 2025): its full-size and 8B tables provide the DeepSeek scores and the original Qwen3-235B-A22B comparison. The full-size HMMT score is 79.4% in that card. R uses ReTool Table 1 (arXiv v1) (Feng et al., 2025), including both the Qwen2.5-32B-Instruct and DeepSeek-R1-Distill-Qwen-32B variants. D also supplies the Gemini-2.5-Flash-Thinking-0520 and o3-mini (medium) comparison rows. DeepSeek uses 16 sampled responses to estimate pass@1 with a 64K generation limit; ReTool averages 32 repeats. For MoE models, the tables report total and active parameter counts separately.

Verified sources. L uses the single-trajectory, Git-hacking-prevented SWE-Lego scores (arXiv v2) (Tao et al., 2026). E uses the original DeepSWE model card (Luo et al., 2025): 42.2% is pass@1 averaged over 16 runs with R2E-Gym, 64K context, and 100 steps. M uses SWE-Master Table 3 (arXiv v1) (Song et al., 2026), whose 61.4% single-attempt result uses Qwen2.5-Coder-32B-Instruct, R2E-Gym, 128K context, and 150 turns. S transcribes historical Qwen3-235B and Claude 3.7 system results compiled in SWE-smith Table 3 (arXiv v1) (Yang et al., 2025b). The selected Verified reference results report single-attempt success rates under the source protocols described above.

Pro sources and evaluation split. P uses Table 1 of the SWE-bench Pro paper (arXiv v2) (Deng et al., 2025), evaluated with SWE-agent on the 731-task public set with augmented problem statements, requirements, and interfaces. Our Pro experiments use the same public dataset version. These results are separate from the commercial split and the later 642-task dataset V2 (Scale AI, 2026).

B Full Round-wise Coding and TIR Results

Tables 4 and 5 report Human, R0, R1, and R2 for each evaluated model–benchmark pair. Stage-wise gains are calculated from these Co-Harness evaluations. Table 6 lists the second-round gains for both Coding and TIR, complementing the shared stage plot in Figure 3.

Table 4: **Co-Harness dual-loop results on coding.** Resolved tasks (%). Human: static human-designed harness; R0: evolved harness without SFT; R1/R2: one/two joint rounds. Δ(R0) and Δ(Hu) denote R2 gains over R0 and Human, in percentage points. Best scores are bold.

Benchmark	Model	Human	R0	R1	R2	Δ(R0)	Δ(Hu)
Verified	Qwen3-8B	5.0	8.4	31.6	**43.0**	+34.6	+38.0
	Qwen3-32B	12.9	19.2	42.2	**61.8**	+42.6	+48.9
Pro	Qwen3-8B	0.2	0.8	14.3	**18.1**	+17.3	+17.9
	Qwen3-32B	3.4	4.8	27.6	**30.6**	+25.8	+27.2

Table 5: **Co-Harness dual-loop results on TIR.** Accuracy (%). Human: static human-designed harness; R0: evolved harness without SFT; R1/R2: one/two joint rounds. Δ(R0) and Δ(Hu) denote R2 gains over R0 and Human, in percentage points. Best scores are bold.

Model	Benchmark	Human	R0	R1	R2	Δ(R0)	Δ(Hu)
Qwen3-8B	AIME24	59.3	63.3	84.0	**86.7**	+23.4	+27.4
	AIME25	51.3	56.9	66.7	**78.3**	+21.4	+27.0
	HMMT25	34.7	39.6	52.3	**62.9**	+23.3	+28.2
Qwen3-32B	AIME24	72.0	76.7	85.3	**87.3**	+10.6	+15.3
	AIME25	61.3	64.9	83.9	**86.3**	+21.4	+25.0
	HMMT25	46.7	49.8	68.7	**77.0**	+27.2	+30.3
Average		**54.2**	**58.5**	**73.5**	**79.8**	**+21.2**	**+25.5**

C Joint-Round Workflow and HarnessCritic Details

Algorithm 1 gives the joint-round execution order. This appendix expands the diagnosis and validation interfaces.

HarnessCritic model. We use GLM-5.3 as HarnessCritic $\mathcal{C}$ to analyze failed trajectories and propose local harness edits. This diagnostic model is distinct from the Qwen3-8B and Qwen3-32B task models whose parameters θ are optimized by SFT. During a harness-search stage, the task-model weights remain fixed while proposed edits are evaluated through rollouts under the candidate and current harnesses. GLM-5.3 generates diagnoses and edit proposals; candidate rollouts determine acceptance under the validation rule.

Table 6: Second-round gains in percentage points. Δ_2 denotes the increment from R1 to R2.

Benchmark	**Model**	**R1**	**R2**	$\mathbf{\Delta_2}$
Verified	Qwen3-8B	31.6	43.0	+11.4
Verified	Qwen3-32B	42.2	61.8	+19.6
Pro	Qwen3-8B	14.3	18.1	+3.8
Pro	Qwen3-32B	27.6	30.6	+3.0
AIME24	Qwen3-8B	84.0	86.7	+2.7
AIME25	Qwen3-8B	66.7	78.3	+11.6
HMMT25	Qwen3-8B	52.3	62.9	+10.6
AIME24	Qwen3-32B	85.3	87.3	+2.0
AIME25	Qwen3-32B	83.9	86.3	+2.4
HMMT25	Qwen3-32B	68.7	77.0	+8.3

C.1 HarnessCritic Taxonomy and Validation Interface

Table 7: HarnessCritic's actionable diagnoses and model-side abstention. The labels guide candidate generation; validation determines whether a proposed edit is retained.

Diagnosis	Component	Example evidence
Prompt ambiguity	P	Conflicting tool-use or submission instructions
Tool-interface error	T	Rejected call, invalid schema, or backend mismatch
Missing skill	S	Absent reusable routine for a recurring operation
Middleware mismatch	W	Retry, turn control, or context handling interrupts progress
Memory failure	M	Persistent state is stale, inaccessible, or over capacity
Model-side error	None	Failure not supported as a harness defect by the trace

Each diagnosis records a root cause, implicated component, severity, trace evidence, and suggested edit. Records labeled `agent_error` are excluded from harness patch generation. The evidence identifies an observed event, while the suggested edit describes a testable intervention. The same symptom, such as repeated tool calls, may reflect a tool-interface defect, an execution-control policy, or a model reasoning error.

Validation sets. The method compares candidate and current harnesses on a targeted validation subset and a broader regression-check subset, holding the model fixed. It accepts an edit only when targeted reward improves and measured reward on the check subset does not decrease. A registry associates each accepted configuration with its parent, edit, evaluation outputs, and decision. Failed or rejected candidates remain diagnostic evidence but do not replace the active configuration.

D Supplementary LongCat Coding Results

D.1 Fixed-Model Harness Optimization

The LongCat study evolves a coding harness with model weights held fixed and compares it to a Claude Code harness (Table 8). Each comparison uses the same LongCat model with the initial, Claude Code, and evolved harnesses.

For LongCat-Flash-Lite (Liu et al., 2026) on Verified, the evolved harness reaches 67.2%, compared with 64.6% using the Claude Code harness. On Pro, the corresponding scores are 29.5%, 35.9%, and 41.8% for the initial, Claude Code, and evolved harnesses. LongCat-Pro-Preview improves from 74.2% to 76.84% on Verified and from 46.3% to 52.4% on Pro.

Relationship to HarnessCritic. Both implementations use trajectory feedback to propose reversible harness changes. The LongCat system uses a research agent editing a workspace organized into six component spaces, together with persistent experiment records. This study evaluates harness optimization with fixed model weights.

Table 8: **Supplementary LongCat harness optimization.** Scores (%) with model weights fixed. ΔCC compares the optimized harness with the Claude Code harness with the same LongCat model held fixed.

Model	Benchmark	Initial	Claude Code	Optimized	ΔCC
LongCat-Flash-Lite	Verified	52.20	64.60	67.20	+2.60
	Pro	29.50	35.90	41.80	+5.90
LongCat-Pro-Preview	Verified	74.20	74.60	76.84	+2.24
	Pro	46.30	49.30	52.40	+3.10

D.2 Independent Model Post-Training Results

Table 9: Independent LongCat-Flash-Lite model post-training results (%). These experiments use independently initialized configurations, evaluated separately from the harness checkpoints in Table 8.

Benchmark	Before training	After training	Gain (pp)
SWE-bench Verified	64.19	67.70	+3.51
SWE-bench Pro	36.80	40.22	+3.42

Table 9 reports independent LongCat model post-training experiments. These experiments use independently initialized configurations; their before/after comparisons are reported separately from the fixed-model harness study.

Figure 5 presents 28 configurations evaluated during the initial harness-search stage with model weights fixed. The selected R0 harness reaches 19.2% before model training.

E Coding Evolution Case

Single-lever exploration. Environment detail, repository mapping, edit reflection, and test hints exceed the 12.9% baseline. Edit reflection is the strongest of these variants at 15.4%. Longer reasoning and the full combination score 12.4% and 12.1%, respectively; adding more instructions does not guarantee an improvement.

Combination and runtime intervention. The first combined variant reaches 15.8%. Combining stall handling with test-error analysis reaches 16.4%. The best configuration adds submission gating, scope checks, and detection of repetitive probes, reaching 19.2%, or 2.8 pp above the preceding best. Probe detection uses sequence similarity at least 0.85 and a soft hint after three similar probes. These mechanisms change runtime control rather than only prompt wording. The reported score measures this combined runtime intervention.

Regression and selection. Later prompt variants score between 12.6% and 15.0%, below the selected configuration. The best-so-far curve therefore remains at 19.2% despite continued exploration. Table 10 records the keep/discard decision for each configuration.

The candidate curve includes unsuccessful alternatives, whereas the staircase tracks the retained best score. The retained harness stays active while alternative candidates are evaluated. Configuration 21 remains selected while configurations 22–27 score between 13.0% and 14.8%. Showing these alternatives exposes the unsuccessful exploration omitted by a summary of improving milestones.

These fixed-model changes are distinct from the joint-round progression above. The complementary TIR case documents recovery, execution efficiency, and rollback under a runtime budget (Sections 5.3 and F). Both searches modify runtimes within supplied task environments.

Complete Verified Configuration Record. Table 10 lists all 28 configurations plotted in Figure 5, including their scores and keep/discard decisions.

Table 10: All 28 Qwen3-32B Verified harness configurations. Scores are resolved-task percentages; status labels indicate keep/discard decisions.

ID	Phase	Configuration	Score (%)	Status
0	0	Baseline	12.9	keep
1	1	opt1 env-detail	13.2	keep
2	1	opt2 repo-map	13.6	keep
3	1	opt3 edit-reflect	15.4	best
4	1	opt4 test-hint	13.2	keep
5	1	opt5 long-CoT	12.4	discard
6	1	full-all	12.1	discard
7	2	combined -v1	15.8	keep
8	2	efficient -tools	14.6	keep
9	2	targeted -replan	15.2	keep
10	2	streamlined -steps	13.4	discard
11	2	progress -stall	13.6	discard
12	2	faster -stall	15.6	keep
13	2	edit-fail -detect	14.8	keep
14	2	submit -verify	14.6	keep
15	2	bounded -thought	12.2	discard
16	2	multi-stall +err	16.4	best
17	3	v2-keyword -test	14.4	discard
18	3	v3-analog -bug	15.0	discard
19	3	v4-TDD	12.6	discard
20	3	v5-dual -search	14.2	discard
21	3	Runtime Intervention	19.2	final
22	3	v6-import -cov	13.0	discard
23	3	v7-full -prompt	13.6	discard
24	3	v8-read -test	13.4	discard
25	3	v9-no -tier3	13.8	discard
26	3	v10-inject -test	13.2	discard
27	3	v11-api -hint	14.8	discard

F TIR Runtime and Extended Evolution Case

Figure 6 expands the configurations in Figure 4 with recorded repair, regression, and rollback milestones. The figure displays the recorded accuracy and runtime values for each milestone.

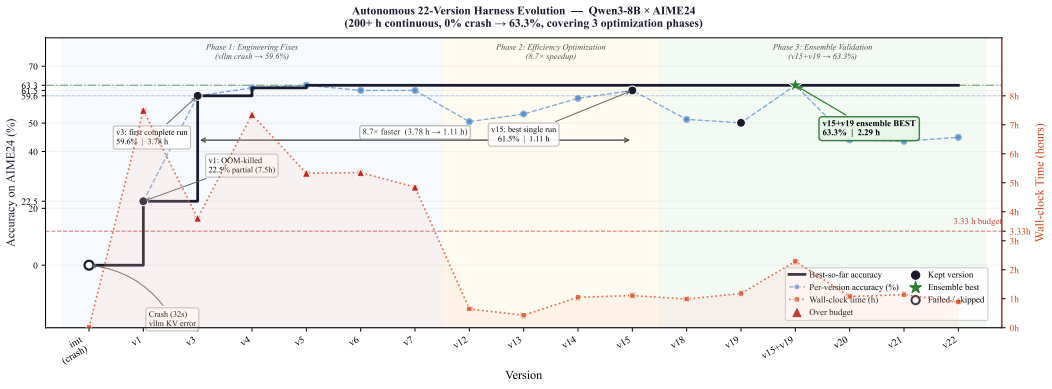

Figure 6: **Recorded milestones in AIME24 harness evolution.** Filled circles show completed single configurations; the hollow point is an incomplete run, and the star is the separate v15+v19 voting ensemble. The crash has no accuracy score. The right axis shows recorded runtime, including time spent on failed runs. Model weights remain fixed.

Interaction loop. The TIR agent alternates generated text with Python interpreter calls. The harness parses a call, dispatches code to a subprocess, and returns execution output or an error as the next observation. The initial configuration uses a 30-second interpreter timeout, up to three retries on tool-call exceptions, truncated interpreter outputs, and no persistent memory or reusable skill library. The case study changes turn limits and execution scheduling during evolution.

Reported evolution. The extended case follows a 22-version, 200+ hour run with Qwen3-8B. Early revisions address an inference-backend configuration error and subprocess cleanup; later revisions explore batching, follow-up-turn reasoning, and ensemble aggregation. Table 11 reports selected configurations. Model weights remain fixed throughout this case.

Table 11: Selected configurations in the autonomous TIR case. The last row combines two seeds by majority voting.

Configuration	Accuracy (%)	Time (h)	Aggregation
First complete run (v3)	59.6	3.78	Single configuration
Batched run (v15)	61.5	1.11	Single configuration
Independent seed (v19)	50.1	1.18	Single configuration
Ensemble (v15 + v19)	63.3	2.29	Six-trajectory vote

The first completed run achieves 59.6% in 3.78 hours. A later batched configuration reaches 61.5% in 1.11 hours, giving a 3.4× runtime ratio between these configurations. Combining it with an independent seed yields a six-trajectory majority-vote score of 63.3% in 2.29 hours, within the stated 3.33-hour budget. Prompt changes that reduced performance were rolled back. The run records recovery from execution failures, faster completed evaluations, and rollback after performance regressions.

Interpretation. The case study reports six-trajectory majority-vote accuracy. Its runtime ratio measures the combined changes to batching, follow-up reasoning, and turn limits between v3 and v15.

Repair and rollback details. The initial 32-second crash results from a `gpu_util` mismatch in the inference backend. A subsequent configuration suffers a zombie-thread/OOM failure after 7.5 hours, with 22.5% recorded before termination. This point is marked as an incomplete run in the figure. Switching from `ThreadPool` to `ProcessPool`, adding process cleanup, and enabling reasoning leads to v3. Versions v4–v7 explore five samples, prompt variants, and temperature schedules; v5 reaches 63.3% but exceeds the time budget. During v12–v15, batching initially reduces accuracy to 50.5% at v12 before reasoning on follow-up turns is restored. The independent v19 run uses seed 42. Domain-specific prompts in v20–v22 cause an approximately 16.5 pp regression and are rolled back.

G Harness Configuration and Attribution Prompt

Table 12 specifies the initial TIR configuration. The harness evolves during optimization; the coding experiments use repository-facing tools.

Tool interaction. The middleware parses calls, executes Python, and appends the tool result to the conversation before the model continues. Tool interactions use the following serialization:

```
<tool_call>
{"name": "python", "arguments": {"code": "..."}}
</tool_call>
<tool_result>
{"stdout": "42\n", "stderr": "", "exit_code": 0}
</tool_result>
```

Attribution prompt. The following template describes the attribution interface for actionable harness failures. Its `Mid` category corresponds to W. Records labeled `agent_error` are excluded from patch generation; the patch fields below describe actionable edits.

Table 12: Initial TIR harness configuration. W denotes middleware, called `Mid` in the attribution interface.

Component	Setting
Prompt (P)	512-token mathematical-reasoning and Python-use instructions; no few-shot retrieval.
Tools (T)	Sandboxed Python subprocess; JSON fields `name: python` and `arguments.code: str`; 30-second timeout.
Skills (S)	Empty initial skill set.
Middleware (W)	Initial maximum 15 turns; terminate on `<DONE>` or final-answer token. Retry tool-call exceptions up to three times with linear backoff. Sliding-window compression retains the last 32k tokens; interpreter outputs are truncated to 2k tokens. The case later increases the turn limit to 30.
Memory (M)	No persistent memory or external retrieval store initially.

```
System: You are an expert agent scaffolding engineer.
Given a failed agent trajectory and the current Harness
configuration, identify the root cause of failure.

Output your reasoning in <think>...</think>, then output
 a JSON object matching this schema:
{
  "root_cause": one of [
    "prompt_ambiguity", "tool_schema_error", "skill_missing",
    "middleware_mismatch", "memory_overflow"
  ],
  "harness_dim": one of ["P", "T", "S", "Mid", "M"],
  "severity": one of ["critical", "major", "minor"],
  "evidence": "<1 sentence citing specific trajectory event>",
  "diff_suggestion": {
    "field_path": "middleware.hooks.post_tool_call[0].type",
    "old_value": "<previous field value>",
    "new_value": "retry_on_rate_limit"
  }
}
== CURRENT HARNESS CONFIGURATION ==
{harness_config_yaml}
== FAILED TRAJECTORY ==
{trajectory_text}
```

Validation and registry. Let $\widehat{J}_{\mathcal{V}}(\theta, \phi)$ denote empirical reward on validation subset $\mathcal{V}$. With the model fixed, the acceptance rule is

$$\delta_{\text{in}} = \widehat{J}_{\mathcal{V}_{\text{in}}}(\theta_t, \widetilde{\phi}) - \widehat{J}_{\mathcal{V}_{\text{in}}}(\theta_t, \phi) > 0, \tag{8}$$

$$\delta_{\text{out}} = \widehat{J}_{\mathcal{V}_{\text{out}}}(\theta_t, \widetilde{\phi}) - \widehat{J}_{\mathcal{V}_{\text{out}}}(\theta_t, \phi) \geq 0. \tag{9}$$

The first subset targets the diagnosed failure; the second covers previously successful or orthogonal behaviors. These quantities correspond to ΔJ_{target} and ΔJ_{check} in Section 3.3. Both accepted and rejected candidates have recorded decisions; only accepted changes replace the active harness. An illustrative layout is:

```
harness_registry/
  v0/harness.yaml
  v1/harness.yaml
     diffs/001_hook_retry.yaml
           002_tool_timeout.yaml
           003_prompt_scope.yaml
  v2/harness.yaml
     diffs/...
```

Each patch records its parent configuration, old and new field values, attribution source, validation differences, and acceptance decision. Version numbers identify harness configurations; R0, R1, and R2 identify optimization stages.