CoEvo-RSI: A Recursive Self-Improvement Framework for Long-Horizon Model–Harness Co-Evolution

Anonymous authors
Paper under double-blind review

Abstract

LLM-based agents now often couple a foundation model with an execution harness that determines what the model observes and how its actions are carried out. Recent work has begun enabling agents to self-improve their own harnesses. However, evolving the harness alone can create a growing mismatch between the model and its execution interface. Recursive co-evolution addresses this asymmetry by iteratively updating both components, yet over long horizons shared trajectory updates can erode model–harness compatibility, while fixed routing rules can become misaligned and cause saturation. We introduce CoEvo-RSI, a Recursive Self-Improvement framework proposed for long-horizon model–harness co-evolution. CoEvo-RSI scores each task for a receiving component based on the null-corrected effect of the counterpart's latest update and a failure attribution weighted by the outcomes of prior interventions. Across 40 rounds on four benchmarks under equal rollout caps, CoEvo-RSI continues to improve while standardized adaptations of published systems plateau or decline, and it finishes ahead of each of them on every benchmark.

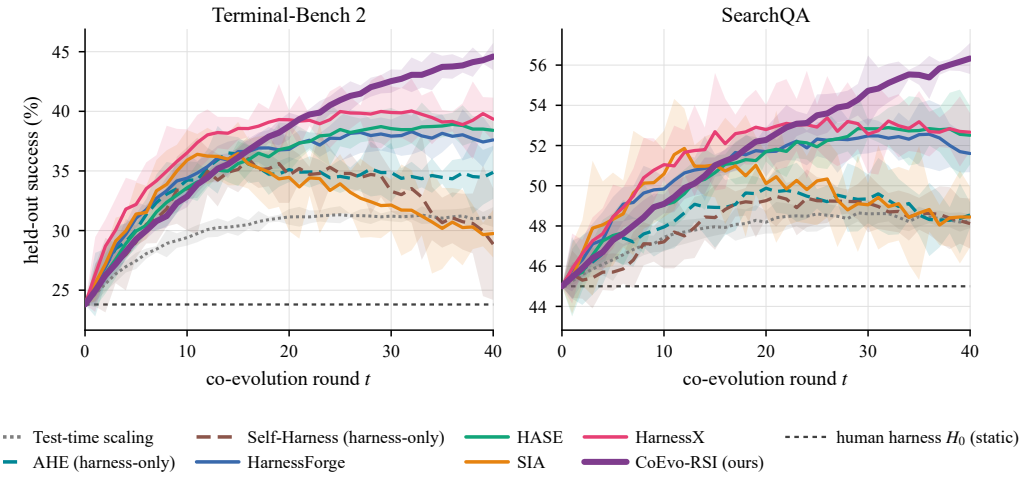


Figure 1: Held-out success over 40 co-evolution rounds on Terminal-Bench 2 and SearchQA for seven comparison arms. CoEvo-RSI starts more slowly than several baselines and overtakes them after roughly round 18.

1 Introduction

LLM-based agents commonly couple a foundation model with an execution layer that mediates its interactions with the environment, referred to as the harness. A harness specifies the system instructions, exposes tools and their implementations, manages memory and context, controls execution and sub-agent delegation (Weng, 2026). As agentic tasks become longer and more interactive, these

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.

components can substantially change what the same underlying model is able to accomplish. This is particularly evident in modern coding agents such as Claude Code and Codex, where much of the system-level capability arises not only from the underlying model, but also from the runtime that decides what information the model sees and how its actions are executed.

A recent line of work removes the human effort from building this harness layer through self-improving harnesses. They allow an agent to inspect its own execution trajectories, diagnose recurring failures, and modify the code or configuration of the harness that produced them (Zhang et al., 2026b; Lin et al., 2026; Elmaaroufi et al., 2026; Shao et al., 2026; Wei et al., 2026). Empirical results from these systems suggest that a frozen language model can acquire substantially stronger agentic capability simply by operating through a better harness. However, an asymmetry in these self-improvement loops is that the harness evolves while the model remains fixed. The resulting harness is therefore optimized for a model that was originally trained under a different execution interface. As evolution continues, this mismatch may become increasingly restrictive, e.g., a new execution procedure may conflict with behaviors learned under the previous environment, or a mechanism that was useful for an earlier model may become unnecessary once the model acquires the corresponding capability itself. Such an inconsistency breaks the loop required for a fully autonomous self-improving system.

Recent work pursues recursive self-improvement by jointly evolving the model and harness (Fan & Huang, 2026; Chen et al., 2026c; Luo et al., 2026; Chen et al., 2026a). In this *recursive co-evolution* paradigm, the model is trained under an updated harness, which is then further improved using the resulting model. A central challenge is sustaining improvement over long alternating optimization. Existing systems either reuse trajectory evidence across both updates or rely on fixed attribution rules. Shared updates can induce conflicting adaptation and erode model–harness compatibility as both components change (Chen et al., 2026b;c; Yu et al., 2026), while fixed routing can become misaligned as failure responsibility shifts over time (Chen et al., 2026a; Luo et al., 2026). Both effects can weaken subsequent updates and hinder long-horizon co-evolution.

We introduce CoEvo-RSI, a framework for long-horizon model–harness co-evolution. At its core is a trajectory-allocation mechanism that determines which experiences should drive each model and harness update as the two components evolve. The mechanism combines two signals, *Behavioral-Change* and *Self-Judgment*. *Behavioral-Change* Δ measures how the counterpart's latest update changes task success and successful execution cost, identifying behavioral shifts that the receiving component can respond to whether performance improves or regresses. *Self-Judgment* b estimates which component is better positioned to address a failed trajectory, giving persistent failures a receiver-specific priority even when their measured behavioral change is small. A task can therefore become informative for the model after a harness update and for the harness after a model update for different reasons. An intervention-feedback coefficient β_t balances these signals over co-evolution. Early in training, counterpart updates often produce substantial behavioral changes, giving Δ a strong allocation signal. As these changes become smaller later in training, Self-Judgment plays a larger role in directing persistent failures to the component most likely to resolve them.

Across 40 rounds on multiple benchmarks, CoEvo-RSI continues improving after several standardized comparison systems plateau or decline (Figure 1). It starts more slowly than some baselines and overtakes them near round 18. These results highlight CoEvo-RSI's unique advantage in sustaining effective long-horizon co-evolution. Our contributions are

- We introduce CoEvo-RSI, a framework for long-horizon model–harness co-evolution that alternates post-training of the model with autonomous evolution of the harness, allowing both components to improve together.
- We introduce a trajectory-allocation mechanism for model–harness co-evolution that scores tasks separately for each component using *Behavioral-Change* and *Self-Judgment*, balanced through intervention feedback. An exact mixture interpretation characterizes how these signals jointly determine trajectory sampling.
- We develop a systematic validation protocol for both allocation signals. Matched model-only and harness-only interventions on randomly sampled failures provide comparative attribution labels, while null correction and measurement-budget sweeps quantify sampling noise in Behavioral-Change. Fixed judges, human annotations, and checkpoint pairings further evaluate the assumptions underlying the allocation mechanism.

- We evaluate CoEvo-RSI across four benchmarks and seven adapted published systems, complemented by native-protocol experiments, paired statistical tests, and detailed compute accounting. Over long-horizon co-evolution, CoEvo-RSI continues to improve while standardized adaptations of existing methods plateau or decline, ultimately outperforming all of them across every benchmark.

2 METHOD

2.1 PROBLEM FORMULATION

Let θ denote the task model parameters, H the executable harness, and $x \sim \mathcal{X}$ a task drawn from the reference distribution. A trajectory $\tau \sim \pi_\theta(\cdot \mid x, H)$ records the model interactions, tool calls, and environment feedback generated while solving x. A deterministic verifier assigns a binary outcome $r(\tau) \in \{0, 1\}$, while $c(\tau)$ denotes the total number of tokens the model generates across the turns of the trajectory. We treat task success as the primary objective while also accounting for execution efficiency, since an improved harness may reduce the cost of successful trajectories. We therefore define a reference utility that combines success with the conditional cost of successful execution

$$f_x(\theta, H) = \mathbb{E}_{\tau \sim \pi_\theta(\cdot \mid x, H)}\left[r(\tau)\right] - \lambda \frac{\mathbb{E}\left[c(\tau) \mid r(\tau) = 1\right]}{c_{\mathrm{ref}}(x)}, \qquad J(\theta, H) = \mathbb{E}_{x \sim \mathcal{X}}\left[f_x(\theta, H)\right], \quad (1)$$

Here $c_{\mathrm{ref}}(x) > 0$ is the mean successful cost of the initial system on x, averaged over eight rollouts measured once before round 1. When none of them succeeds, the trajectory cap of 120,000 generated tokens supplies the reference. For a state with zero success probability, the conditional cost in f_x is also defined as the trajectory cap. The parameter λ sets the cost trade-off on this fixed task scale. The model update uses binary reward $R(\tau) = r(\tau)$, while cost enters allocation and intervention feedback. Thus J describes system behavior on the reference task distribution, while the model update trains on tasks selected by the allocation rule.

Given (θ_t, H_t), the loop alternates

$$\theta_{t+1} = \mathrm{RL}\left(\theta_t \mid H_t,\, \mathcal{T}_t^{\mathrm{model}}\right), \quad (2)$$

$$H_{t+1} = \mathcal{E}\left(H_t \mid \theta_{t+1},\, \mathcal{T}_t^{\mathrm{harness}}\right), \quad (3)$$

The model update holds H_t fixed and trains on $\mathcal{T}_t^{\mathrm{model}}$. The subsequent harness update holds θ_{t+1} fixed while a coding-agent evolver $\mathcal{E}$ inspects $\mathcal{T}_t^{\mathrm{harness}}$ and modifies the current harness source. These sets represent the tasks allocated to each component. Because the available rollouts exceed both the model's mini-batch budget and the evolver's fixed context, the allocation determines which evidence informs each update. Its priorities should adapt as the system evolves, including when a harness edit creates a new execution path, model training changes tool-use behavior, or an acquired model capability makes an existing harness mechanism redundant (Chen et al., 2026a; Luo et al., 2026). We study trajectory allocation within this fixed alternating update loop.

2.2 COUNTERPART-AWARE ALLOCATION

For a receiver $a \in \{\text{model}, \text{harness}\}$, let $\bar{a}$ denote its counterpart. The quantity $\Delta^{\bar{a}}(x)$ measures how the latest update of $\bar{a}$ changed task x, and $b(x)$ is the component predicted to benefit from receiving a failed task. We combine them in the receiver-specific score

$$w_a(x) = \Delta^{\bar{a}}(x) + \beta_t \mathbb{1}[b(x) = a]. \quad (4)$$

The two signals cover complementary cases. Behavioral-Change can prioritize a task after a counterpart update even when it succeeds, while Self-Judgment preserves failed tasks whose measured behavior changes little. When both apply, their contributions are combined, with the attribution indicator directing Self-Judgment weight to the identified component. The model therefore follows changes induced by harness edits and model-attributed failures, while the harness follows changes induced by model training and harness-attributed failures. Normalizing Eq. 4 for model sampling yields an exact mixture of change-proportional and attribution-based sampling, with attribution share $\alpha_t = \beta_t M_t / (D_t + \beta_t M_t)$ when both terms have positive mass. Here $D_t = \sum_x \Delta^{\mathrm{H}}_{t-1}(x)$ and $M_t = \sum_x \mathbb{1}[b(x) = \text{model}]$ sum over the candidate tasks. The mixture components normalise these two terms separately. The harness uses the same score for Top-K selection. Matched-intervention calibration and sensitivity analyses are provided in Appendices N and C.

2.3 MEASURING BEHAVIORAL-CHANGE

For each task x and system state $S = (\theta, H)$, we draw N rollouts at the training sampling temperature. The empirical success rate is $p_S(x) = N^{-1} \sum_{i=1}^{N} r(\tau_i)$, the success count is $s_S(x) = \sum_{i=1}^{N} r(\tau_i)$, and $c_S(x)$ is the mean generated-token count among successful rollouts. We set $c_S(x)$ to the trajectory cap when $s_S(x) = 0$, solely to keep the expression defined. Measurements compare adjacent states S^- and S^+ that differ in exactly one component. With $\bar{p}(x) = \frac{1}{2}\big(p_{S^-}(x) + p_{S^+}(x)\big)$, let $e_0(\bar{p})$ be the mean of $|p_{S^+} - p_{S^-}|$ for two independent samples of N rollouts with success probability $\bar{p}$ (Appendix C.3). The task score is

$$\Delta(x; S^- \to S^+) = \max\big\{0,\ |p_{S^+}(x) - p_{S^-}(x)| - e_0\big(\bar{p}(x)\big)\big\} + \lambda\, \rho_c\, \frac{|c_{S^+}(x) - c_{S^-}(x)|}{c_{\mathrm{ref}}(x)}. \quad (5)$$

The fixed denominator $c_{\mathrm{ref}}(x)$ keeps the cost component on the same task scale across rounds. The factor $\rho_c = \min(s_{S^-}, s_{S^+})/(\min(s_{S^-}, s_{S^+}) + 1)$ reduces the influence of a cost difference when either state has few successes. It equals 0 when either state has no successful rollout, so the score then uses success change alone, equals $1/2$ at a single success, and approaches one as the counts grow. The round measures model-induced change $\Delta_t^{\mathrm{M}}(x) = \Delta(x; (\theta_t, H_t) \to (\theta_{t+1}, H_t))$ for the next harness decision and harness-induced change $\Delta_t^{\mathrm{H}}(x) = \Delta(x; (\theta_{t+1}, H_t) \to (\theta_{t+1}, H_{t+1}))$ for the next model decision. The comparisons isolate which component changed while holding its counterpart fixed.

Magnitude gives priority to improvements and regressions. A harness edit that changes success identifies tasks for the next model update. A model update that regresses a task can reveal a harness assumption for the evolver to revisit. By the triangle inequality, the uncorrected magnitude Δ^{raw}, which omits e_0 from Eq. 5, bounds the absolute signed feedback gain defined below, $|\hat{G}| \le \Delta^{\mathrm{raw}}$, and success and cost effects can cancel in the gain while still revealing a substantial change in behavior. Appendix L compares $N \in \{4, 8, 16, 32\}$ under the same per-round cap.

2.4 SELF-JUDGMENT AND VALIDATED RELIABILITY

For each failed trajectory, a separate post-execution call asks the task model which component is more likely to improve the outcome. It sees the task statement, full trajectory, and tool manifest, then returns $b(\tau) \in \{\text{harness}, \text{model}\}$ (Appendix H). The call does not alter the execution trajectory or enter policy training. We take a majority vote over failed trajectories for each task, with ties assigned to the model, and set $b(x) = \varnothing$ when all rollouts succeed. The label supplies a preference for a receiver, and each receiver uses the label from the latest measurement before its own update, which is the first measurement of the round for the model and the second for the harness (Appendix D). Its weight follows intervention feedback through

$$\beta_t = \beta_{\max} \max\left(0,\ 2q_{t-1} - 1\right), \quad (6)$$

where q_t is the fraction of previously routed interventions whose intervention gain

$$\hat{G}(x; S^- \to S^+) = \big(p_{S^+}(x) - p_{S^-}(x)\big) - \lambda\, \rho_c\, \frac{c_{S^+}(x) - c_{S^-}(x)}{c_{\mathrm{ref}}(x)} \quad (7)$$

was positive, with $q_0 = 1/2$ and the previous value retained when no intervention became observable. Each routed task occurrence counts once, model-side and harness-side outcomes are pooled, an outcome is assigned to the round in which it is measured, and q_t averages the outcomes observed in rounds $t - 2$ to t (Appendix D). The gain uses the same reference cost scale as Eq. 1, with the reliability factor ρ_c applied to its empirical cost difference. A positive gain therefore denotes an improvement under the feedback statistic used to set β_t.

The mapping in Eq. 6 gives zero attribution weight when observed intervention success is at or below one half and raises the weight as that rate increases. The threshold is a choice for controlling how strongly the router trusts its own judgments. In the main experiments, $\beta_{\max} = 0.5$. A weak judge consequently leaves allocation close to change-based routing, while successful interventions allow failed tasks assigned to a receiver to receive more attention.

The feedback q_t is measured on tasks selected for an intervention by the router. It records whether the chosen update improved reference utility, while the alternative update on that task is usually unobserved. Its value therefore controls the attribution weight without serving as a direct measure of

comparative attribution accuracy. To measure that accuracy, each round we draw 16 failed tasks independently of routing. From the same starting state, each receives a small model-only update and a harness-only edit on discarded branches. Their gains yield a comparative label $\ell(x)$, with near-equal gains treated as ties (Appendix N). Comparing b with ℓ gives calibrated accuracy and a prevalence-independent Youden index J_t^{cal} (Youden, 1950). We evaluate the variant $\beta_t = \beta_{\max} \max(0, J_{t-1}^{\mathrm{cal}})$ alongside Eq. 6. These calibration interventions use 2.3% of the round budget.

2.5 Allocation and the Round

The candidate panel is sampled uniformly from the evolution pool before either receiver scores it. Panels rotate across rounds, giving tasks outside the current panel future opportunities to enter both update distributions. The panel contains 44 tasks on Terminal-Bench 2, 65 on GAIA, and 184 on SWE-bench Pro and SearchQA, and the evolver receives $K = 16$ of them. Measurements, calibration interventions, edit validation, and model training all draw from the same per-round rollout cap of 16,384. The selected tasks therefore affect the evidence seen by the next update without changing the total execution allowance. For the model, normalising w_{model} defines $P_t^{\mathrm{model}}(x) \propto w_{\mathrm{model}}(x)$, from which training mini-batches are drawn. Sampling is uniform if all weights vanish, as in round 1 where no harness update has occurred, and the model update uses no importance correction. For the harness, $\mathcal{T}_t^{\mathrm{harness}} = \mathrm{TopK}(w_{\mathrm{harness}})$ supplies the evolver with K tasks, their full trajectories, and the current source. The evolver may patch instructions, tools, memory and context management, or control flow. An edit is committed after a path guard, safety linter, compilation, unit tests, and a fixed regression suite pass. In one round, the model consumes the preceding harness-induced change $\Delta_{t-1}^{\mathrm{H}}$, trains, and yields a model-induced change Δ_t^{M} for the harness update. Calibration and observable intervention outcomes then refresh β for the next round. Adjacent comparisons share an endpoint and the state reached at the end of the preceding round is reused for tasks that remain in the panel, so a round needs two fresh measurement states for those tasks and a third for the quarter of the panel that rotates in. Algorithm 1 in Appendix D specifies the recursion, and Table 4 accounts for measurement, training, validation, and calibration rollouts.

3 Experiments

3.1 Setup

We evaluate $T = 40$ rounds on Terminal-Bench 2 (Merrill et al., 2026), SWE-bench Pro (Jimenez et al., 2024; Deng et al., 2025), GAIA (Mialon et al., 2024), and SearchQA, which combines HotpotQA (Yang et al., 2018) with 2WikiMultiHopQA (Ho et al., 2020). Terminal-Bench 2 uses 312 disjoint evolution tasks and the pinned 2.0 release of 89 held-out tasks. SWE-bench Pro evolves on SWE-Gym (Pan et al., 2024) and holds out the public set. GAIA uses a 65/100 validation split and one submission per system, from its first seed, to the private 300-task test set. SearchQA uses 2,000 evolution and 1,000 held-out tasks. Every arm uses one runtime, starts from the strongest compatible human harness H_0 (Terminus-2, OpenHands, OWL, or Search-o1), and has the same editable components. Appendix D names the other human harnesses and the backbones. Unless noted, Qwen3.5-35B-A3B serves as task model, judge, and evolver. Policy optimisation uses GRPO (Shao et al., 2024), with group size 8, $K = 16$ evolver tasks, $\lambda = 0.2$, $\beta_{\max} = 0.5$, and $N = 8$ measurement rollouts per state, fixed from a pilot. The per-round cap is 16,384 rollouts for measurement, calibration, validation, and training, yielding 12–14 GRPO iterations. Appendix F reports tokens, GPU-hours, evolver calls, judge calls, and a control with an enlarged rollout allowance.

We compare standardized adaptations of HarnessX, HarnessForge, HASE, SIA, and HELIX (Fan & Huang, 2026), the harness-only systems AHE (Lin et al., 2026) and Self-Harness (Zhang et al., 2026a), three static human harnesses, and budget-matched test-time scaling. The adaptations share the runtime, evolver backbone, optimiser, cap, and split, and keep the update schedules, gates, and candidate generation that Appendix E lists for each system. Section 3.4 isolates allocation by placing their allocation rules in one identical loop. Appendix E documents the changes and reports native-protocol runs. We report final score y_{40}, drawdown from peak $D_{\mathrm{pf}} = \max_t y_t - y_{40} \geq 0$, second-half slope, gain after round 15, saturation round t_{95}, round-to-round variability σ_Δ, area under the curve, and last-five-round mean. These are correlated summaries of the same trajectory. Main results use five seeds, and appendix studies use three unless noted. Curves show 95% intervals of the seed

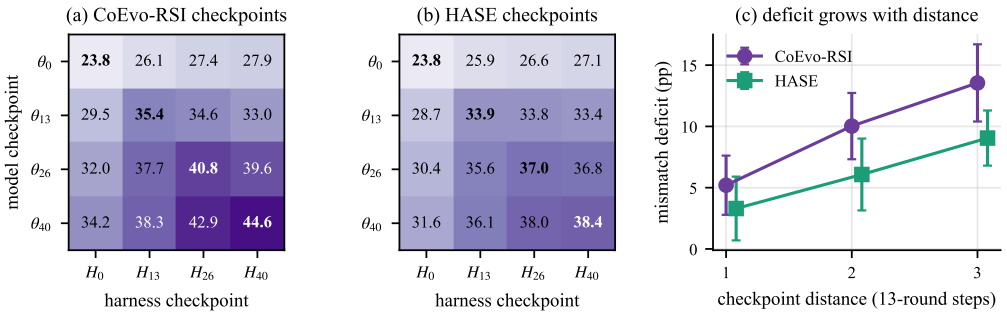


Figure 2: Model–harness compatibility on Terminal-Bench 2. (a, b) Held-out success for every pairing of model and harness checkpoints from one run (three seeds). Diagonal cells are states the run visited. (c) Deficit of a mismatched pair relative to the better of its two matched pairs, by checkpoint distance.

mean. Paired task-level bootstrap intervals and permutation tests are clustered by repository on SWE-bench Pro (Table 8).

3.2 Model–Harness Mismatch

Figure 2(a) examines compatibility between model and harness checkpoints taken from different stages of the same co-evolution run. Both components evolve here, allowing us to directly test whether improvements remain compatible across training stages. Along the diagonal, where model and harness checkpoints are matched, held-out success increases from 23.8 at round 0 to 35.4, 40.8, and 44.6 at rounds 13, 26, and 40, respectively. Cross-checkpoint pairings reveal a different pattern. For θ_{13}, replacing its matched harness H_{13} with later harnesses H_{26} or H_{40} reduces success from 35.4 to 34.6 and 33.0. Likewise, θ_{26} achieves 40.8 with H_{26} but only 39.6 with H_{40}. These regressions show that later harnesses are not uniformly beneficial to earlier models. As co-evolution progresses, the two components become increasingly specialized to the behavior of their contemporaneous counterpart.

The columns reveal the complementary effect. Later models generally improve performance under a fixed harness, while the final model continues to benefit from concurrent harness evolution. For θ_{40}, success rises from 34.2 with H_0 to 44.6 with H_{40}. Figure 2(b) shows the same qualitative pattern for HASE, although the gap between matched and mismatched checkpoints is smaller. Figure 2(c) summarizes this effect using the mismatch deficit relative to the better of the two corresponding matched states. For CoEvo-RSI, the deficit increases from 5.2 points for adjacent checkpoints to 13.6 points for the most distant pairing. The important observation is that this effect cannot be explained by ordinary training progress alone. Later models often perform better under earlier harnesses, yet earlier models can regress under later harnesses. The incompatibility is therefore directional and reflects how each component adapts to the evolving behavior of the other. This motivates counterpart-aware trajectory allocation in CoEvo-RSI as an experience that becomes informative after a model update need not remain equally informative after a harness update, and vice versa.

3.3 Long-Horizon Co-Evolution

Figure 1 highlights the main advantage of CoEvo-RSI in long-horizon co-evolution. CoEvo-RSI does not rely on the fastest initial gains. Several alternatives lead during the early rounds on Terminal-Bench 2 and SearchQA, before CoEvo-RSI overtakes them around round 18 and continues improving through round 40. In contrast, SIA and Self-Harness gradually lose part of their early gains, while HarnessForge and HASE remain more stable but plateau earlier. The separation therefore emerges primarily in the second half of co-evolution, showing that CoEvo-RSI is particularly effective at sustaining improvement as the model and harness undergo repeated mutual adaptation.

Table 1 confirms this long-horizon advantage across all four benchmarks. At round 40, CoEvo-RSI reaches 44.6 on Terminal-Bench 2, 56.0 on SWE-bench Pro, 43.5 on GAIA, and 56.3 on SearchQA,

Table 1: Final held-out success at round 40 and drawdown from peak D_{pf}. Underlined values are the strongest baseline per column.

	Terminal-Bench 2		SWE-bench Pro		GAIA		SearchQA		Avg.
Method	final	D_{pf}	final	D_{pf}	final	D_{pf}	final	D_{pf}	final
Human-engineered harnesses (static, no evolution)									
H_0: best human harness†	23.8	–	44.0	–	27.0	–	45.0	–	35.0
second human harness†	22.6	–	42.1	–	25.9	–	43.1	–	33.4
third human harness†	21.0	–	40.3	–	24.2	–	41.6	–	31.8
Evolving systems, standardized adaptations (40 rounds, 5 seeds)									
Test-time scaling	31.1	0.2	49.0	0.2	30.7	0.2	48.2	0.4	39.8
AHE (harness-only)	34.9	1.6	49.0	1.3	30.8	2.7	48.5	1.3	40.8
Self-Harness (harness-only)	28.9	7.2	46.4	3.5	24.3	9.3	48.1	1.4	36.9
HarnessForge	37.6	0.7	52.4	0.1	35.6	1.7	51.6	1.0	44.3
HASE	38.4	0.5	51.8	1.3	37.1	1.2	52.5	0.4	44.9
SIA	29.7	6.7	46.1	4.2	27.9	6.5	48.4	3.4	38.1
HarnessX	39.3	0.7	46.3	4.6	26.7	11.3	52.7	0.7	41.2
HELIX-A (iterated)	40.1	0.9	52.9	0.4	36.2	2.0	53.0	0.6	45.6
CoEvo-RSI (ours)	**44.6**	**0.0**	**56.0**	**0.2**	**43.5**	**0.0**	**56.3**	**0.0**	**50.1**

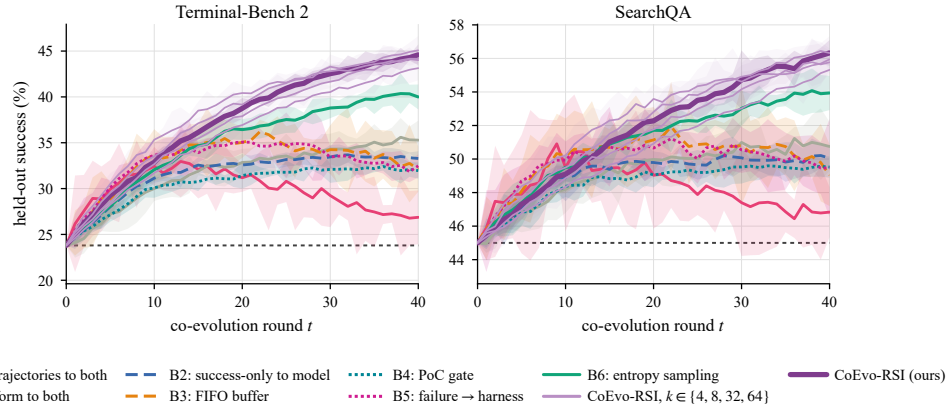


Figure 3: Allocation strategies under an identical loop, evolver, optimiser, cap, and split. Thin curves are CoEvo-RSI with $k \in \{4, 8, 32, 64\}$ policy-update iterations per harness update at a constant rollout count, alongside the default configuration of at most 16 iterations.

exceeding the strongest baseline in each setting by 4.5, 3.1, 6.4, and 3.3 points. Its mean final score reaches 50.1, compared with 45.6 for HELIX-A and 35.0 for the initial human harness. The strongest baseline varies across benchmarks, yet CoEvo-RSI finishes ahead in every setting, indicating that its long-horizon gains are consistent across different tasks.

The drawdown results further show that CoEvo-RSI preserves the progress accumulated over long runs. It finishes at its peak on three benchmarks and only 0.2 points below its peak on SWE-bench Pro. Self-Harness, by comparison, loses 7.2 points from its peak on Terminal-Bench 2 and 9.3 on GAIA, while HarnessX loses 11.3 on GAIA. HarnessForge, HASE, and HELIX-A exhibit smaller drawdowns but converge to lower final performance. Together, these results show that CoEvo-RSI is particularly suited to long-horizon model–harness co-evolution, where sustained adaptation and retention of accumulated gains become increasingly important over repeated update rounds.

3.4 Comparing Trajectory-Allocation Strategies

Figure 3 isolates trajectory allocation by keeping the update loop, evolver, optimiser, rollout cap, and split fixed. The seven alternatives shown are uniform sampling (B0), all trajectories to both receivers

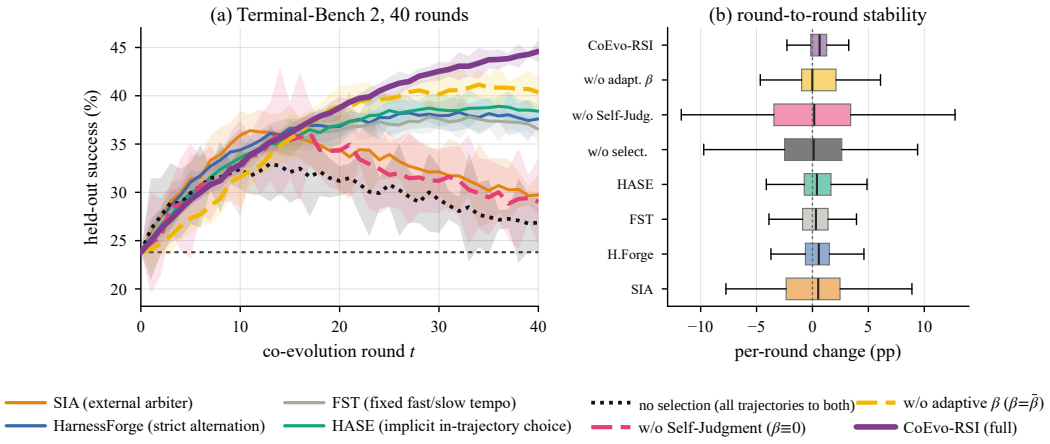


Figure 4: (a) Four published paradigms for balancing model and harness updates and three ablations of CoEvo-RSI on Terminal-Bench 2. (b) Distribution of round-to-round changes in held-out success, pooled over seeds. Box width is the arm's training stability.

(B1), success filtering (B2, HarnessForge), FIFO replay (B3, HarnessX), a proof-of-concept gate (B4, HASE), static failure-to-harness routing (B5), and entropy sampling (B6). On Terminal-Bench 2, sending all trajectories to both receivers initially improves performance, but the curve peaks near round 13 at 33.0 and falls to 26.9. FIFO replay also gains quickly before declining. These trajectories show that supplying more experience, or retaining recent experience, does not by itself maintain useful update priorities as the model and harness change.

More selective rules reduce some of this degradation but still show limited long-horizon progress. Success filtering and the proof-of-concept gate finish at 33.3 and 32.0 on Terminal-Bench 2. Static failure routing improves initially but declines after approximately round 20, highlighting the limitation of assigning fixed receivers as the model and harness continue to evolve. Entropy sampling is the strongest plotted alternative, reaching 40.0 on Terminal-Bench 2 and 53.9 on SearchQA. CoEvo-RSI reaches 44.6 and 56.3, with its advantage expanding in later rounds to 4.6 and 2.4 points, respectively. At $N = 8$, null correction lowers the late-round correlation of change with $\bar{p}(1 - \bar{p})$ from 0.44 to 0.15, and entropy sampling with the same Self-Judgment term reaches 42.0 (Appendix L). The individual contributions of Behavioral-Change and Self-Judgment are examined separately in the ablations. The thin curves vary the number of policy-update iterations per harness update over $k \in \{4, 8, 32, 64\}$, alongside the default of at most 16 iterations, while holding the training rollout count constant at the default allocation. Their final scores span 1.9 points on Terminal-Bench 2 and 1.0 on SearchQA, and they retain the same late improvement pattern. The advantage is therefore present across the tested update ratios and does not depend on a single model–harness tempo.

3.5 Component Ablations and Training Stability

Figure 4(a) compares the full allocation rule with three ablations and four balancing paradigms, including SIA's external arbiter, HarnessForge's strict alternation, FST's fixed tempo (Tiwari et al., 2026), and HASE's within-trajectory selection. Sending all trajectories to both receivers reduces the final score to 26.9, while retaining Behavioral-Change without Self-Judgment ($\beta \equiv 0$) reaches 29.0 and produces a visibly less stable trajectory. These results show that change-based prioritization alone does not sustain improvement in this setting, while receiver-specific prioritization of failed tasks provides an important complementary signal.

Replacing the adaptive coefficient with its time average, $\bar{\beta} = 0.18$, yields a more stable trajectory than removing attribution but reaches only 40.4, compared with 44.6 for the full method. This result shows that adapting the attribution weight over co-evolution provides a clear advantage over a constant weight. Among the four balancing paradigms, HASE performs best at 38.4. CoEvo-RSI trails HASE by 0.4 points at round 10 but leads by 6.2 points at round 40, further showing that its advantage emerges through sustained improvement over longer co-evolution horizons.

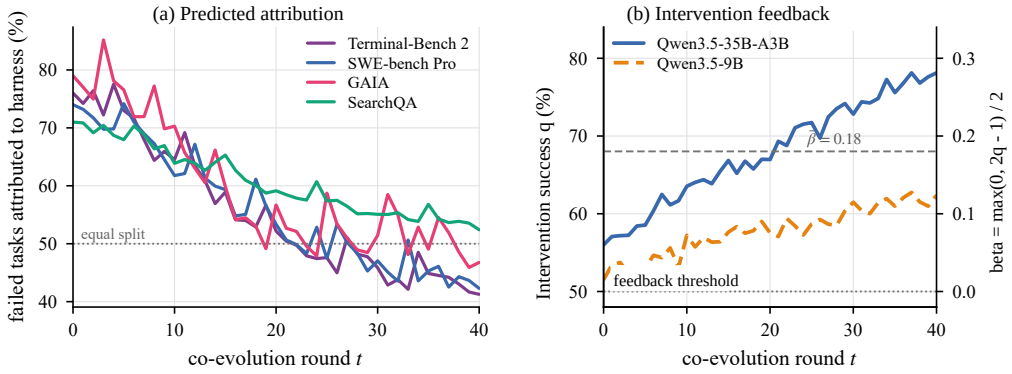


Figure 5: (a) Fraction of failed held-out tasks whose *Self-Judgment* points at the harness, over 40 rounds. (b) Executed-intervention success q_t and the coefficient $\beta_{\max} \max(0, 2q_t - 1)$ it sets for the following round, for a strong and a weak backbone.

Figure 4(b) complements the endpoint comparison by showing the distribution of per-round changes. The no-selection and no-Self-Judgment variants exhibit broader distributions with larger negative changes, while the full method concentrates more tightly around small positive gains. The constant-weight variant falls between these cases. Together, the two panels show that the full allocation rule achieves a higher final score while reducing disruptive updates that can erase earlier progress.

3.6 Predicted Attribution over Rounds

Figure 5(a) tracks how Self-Judgment evolves during co-evolution. The share of failed held-out tasks attributed to the harness decreases from 76.0% to 41.3% on Terminal-Bench 2, from 74.0% to 42.3% on SWE-bench Pro, and from 79.0% to 46.8% on GAIA. The judge therefore shifts from predominantly harness-attributed failures early in training toward more model-attributed failures by round 40. SearchQA shows a smaller shift and remains slightly above an equal split. Local fluctuations indicate that this transition occurs gradually as the remaining failures evolve. These values represent predicted attributions, while their comparative accuracy is evaluated separately through matched interventions.

Panel (b) shows how intervention feedback adjusts the influence of these predictions. For Qwen3.5-35B-A3B, intervention success q_t increases from 56.0% at its first observation to 78.1%, raising the coefficient it sets for the following round from 0.06 to 0.28. The initial value $q_0 = 1/2$ gives $\beta_1 = 0$, so attribution carries no weight in the first round. The adaptive rule therefore assigns less attribution weight than the constant reference $\bar{\beta} = 0.18$ early and more weight later. For Qwen3.5-9B, q_t remains between approximately 52% and 62%, keeping β_t below 0.13. Together, the panels show two forms of adaptation as the predicted receiver evolves with the remaining failures and its influence changes according to observed intervention outcomes.

4 Conclusion

CoEvo-RSI introduces a framework for long-horizon model–harness co-evolution, centered on a trajectory-allocation mechanism that determines which experiences should drive model and harness updates as both components evolve. By combining Behavioral-Change with receiver-specific Self-Judgment through an adaptive intervention-feedback coefficient, CoEvo-RSI shifts from responding to counterpart-induced behavioral changes toward resolving persistent failures as co-evolution progresses. Under equal rollout caps, CoEvo-RSI achieves the highest final score on all four benchmarks, with little or no drawdown, and continues improving throughout the measured 40-round horizon. Controlled allocation comparisons and component ablations connect these mechanisms to the observed gains, while routing, transfer, horizon, and harness analyses show that the improvements persist across settings. Together, these results demonstrate the value of CoEvo-RSI for sustaining improvement over long-horizon model–harness co-evolution.

USE OF AI ASSISTANCE

Language models assisted throughout the preparation of this manuscript, including drafting and editing the text.

REFERENCES

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. Harnessforge: Joint harness and policy evolution for adaptive agent systems. *arXiv preprint arXiv:2606.01779*, 2026a.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. HarnessX: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026b.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-harness: Co-evolving harnesses and model weights for LLM agents. *arXiv preprint arXiv:2607.22688*, 2026c.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Bradley Efron, Robert J Tibshirani, et al. *An introduction to the bootstrap*. Boca Raton, Florida, 2000.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026.

Tianyu Fan and Chao Huang. HELIX: Model-harness co-evolution for recursive self-improvement. *arXiv preprint arXiv:2608.13951*, 2026.

Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

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.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *International Conference on Computational Linguistics (COLING)*, 2020.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. OWL: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025a.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025b.

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 *International Conference on Learning Representations*, 2024.

Laude Institute. Terminus-2: Harbor's reference agent implementation. Harbor framework documentation, 2026. URL `https://www.harborframework.com/docs/agents/terminus-2`.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5420–5438, 2025.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Haochen Luo, Yi Huang, Sichun Luo, Fengyuan Liu, Lei Li, Zefa Hu, Junlan Feng, and Qi Liu. Harness-Aware Self-Evolving: Co-Evolving Model Weights, Harness, and Task Solutions. *arXiv preprint arXiv:2607.03935*, 2026.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *International Conference on Learning Representations*, 2026:40903–40986, 2026.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. *arXiv preprint arXiv:2412.21139*, 2024.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 'smolagents': a smol library to build great agentic systems. `https://github.com/huggingface/smolagents`, 2025.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-R1: Learning to edit executable runtime harnesses from agent failure trajectories. *arXiv preprint arXiv:2608.02276*, 2026.

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

Terminal-Bench Team. Terminal-bench 2.1. Release note, May 2026, 2026. URL `https://www.tbench.ai/news/terminal-bench-2-1`.

Rishabh Tiwari, Kusha Sareen, Lakshya A Agrawal, Joseph E. Gonzalez, Matei Zaharia, Kurt Keutzer, Inderjit S Dhillon, Rishabh Agarwal, and Devvrit Khatri. Learning, Fast and Slow: Towards LLMs That Adapt Continually. *arXiv preprint arXiv:2605.12484*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *International Conference on Learning Representations*, 2025: 65882–65919, 2025.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026.

Lilian Weng. Harness engineering for self-improvement. Lil'Log, 2026. URL `https://lilianweng.github.io/posts/2026-07-04-harness/`.

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*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

W. J. Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950.

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

Zhiyuan Zhai and Xin Wang. Selective rollout: Mid-trajectory termination for multi-sample agent RL. *arXiv preprint arXiv:2605.05802*, 2026.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-Harness: Harnesses That Improve Themselves. *arXiv preprint arXiv:2606.09498*, 2026a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026b.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026c.

APPENDIX CONTENTS

A Related Work and Positioning **15**

A.1 Agent Harnesses 15

A.2 Model–Harness Co-Evolution 15

B Validation and Control Studies **15**

C Analysis of the Allocation Rule **16**

C.1 An Exact Mixture Interpretation 16

C.2 Coefficient and Ranking Sensitivity 17

C.3 Resolution of Behavioral Change and the Null-Corrected Score 17

C.4 Relation between the Routing Score and the Reference Utility 18

C.5 Prevalence-Aware Reliability and Selection Bias 18

C.6 The Model's Training Distribution 18

D Protocol Details **19**

E Baseline Protocols and Change Manifest **22**

F Compute Ledger and Budget Matching **23**

G Task-Level Statistics and Hidden-Set Evaluation **25**

H Prompts **25**

H.1 Task agent 25

H.2 Self-judgment 26

H.3 Harness evolver 27

H.4 Baseline-specific calls 28

I Additional Co-Evolution Curves **29**

J Detailed Metrics **29**

J.1 Improvement-Trajectory Metrics 29

J.2 Predicted-Attribution Details 30

J.3 Allocation-Rule Details 31

J.4 Balancing-Paradigm Details 31

K β Schedules, Feedback Controls, and Negative Controls **31**

L Measurement Budget, Null Correction, and Uncertainty Sampling **33**

M Signed Change and Same-Side Repair **34**

N Attribution Calibration with Matched Interventions and Judge Controls **34**

O Model–Harness Compatibility Matrix **36**

P Different Model Scales and Initial Harnesses **36**

Q The Allocation Rule inside Other Systems **38**

R Factorial Transfer of Model and Harness **39**

S Beyond Forty Rounds **40**

T Harness Complexity, Gate Outcomes, and Safety Audit **40**

U Case Studies **41**

U.1 Case 1. A harness edit creates work for the model 41

U.2 Case 2. A model update makes part of the harness obsolete 43

U.3 Case 3. Self-Judgment recovers missed tasks 45

U.4 Case 4. Adaptive weighting absorbs an incorrect attribution 46

U.5 Case 5. Predicted responsibility shifts within a task family 47

U.6 Case 6. The harness sheds an obsolete component 48

V Sensitivity to λ and K **51**

W Limitations **51**

A Related Work and Positioning

A.1 Agent Harnesses

LLM agents are typically deployed through a system surrounding a base model that decides how it thinks and plans, calls tools and acts, and manages context. Recent usage calls this system the harness (Weng, 2026). It goes beyond the classical policy–memory–tools stack by encoding workflow structure and persistent state, so that the same underlying model can occupy execution interfaces of very different power (Yao et al., 2023; Roucher et al., 2025; Wang et al., 2025; Yang et al., 2024; Li et al., 2025; Hu et al., 2025a). A recent line of work removes the human from this design loop by having an agent inspect its own execution trajectories and edit the harness that produced them. Context-engineering methods keep the model frozen and evolve only what it is shown. Zhang et al. (2026c) maintain a structured, incrementally curated context playbook and avoid rewriting the prompt outright. Workflow-search methods treat the orchestration graph itself as the object to optimise, using a meta-agent to propose new workflows against an archive of prior designs (Hu et al., 2025b) or a search over a graph of nodes and edges (Zhang et al., 2024). A third group edits the harness source directly with a coding agent, inspecting failures and committing patches against a regression gate (Zhang et al., 2026b; Elmaaroufi et al., 2026; Shao et al., 2026; Wei et al., 2026; Lin et al., 2026; Zhang et al., 2026a). Lin et al. (2026) additionally attach a prediction to every edit and later check it against task outcomes (decision observability), and report that the capacity to produce a useful edit is comparatively stable across backbones of very different scale. CoEvo-RSI's harness evolver follows the same inspect–diagnose–patch loop as this line of work.

A.2 Model–Harness Co-Evolution

Joint optimisation extends harness evolution to include policy updates recently present in several systems. Fan & Huang (2026) formulate model–harness co-evolution as a build–update–rebuild cycle over a source-traceable substrate and harvest verified records from harness runs as model-training signal. Chen et al. (2026a) combine fault-guided harness tailoring with harness-conditioned policy alignment. Luo et al. (2026) let a single model produce task solutions and edit harness components through a unified action space. Chen et al. (2026b) assemble typed harness primitives through a substitution algebra, adapt them from execution feedback, and export trajectories as training signal. Hebbar et al. (2026) use a feedback agent to choose between harness and weight updates, and Tiwari et al. (2026) interleave reinforcement learning with prompt evolution on a fixed fast/slow tempo. Zhang et al. (2026a) study autonomous harness improvement through weakness mining, proposal generation, and regression testing. These systems differ in what they edit and how they gate updates, but they share a common property that the trajectories used for each update are selected according to a fixed rule, such as success filtering, recency, a proof-of-concept gate, an arbiter call, or verified-record harvesting. This fixed selection mechanism can limit their effectiveness over long-horizon co-evolution. CoEvo-RSI instead replaces fixed selection with an adaptive update mechanism designed specifically for sustained long-horizon co-evolution.

B Validation and Control Studies

Table 2 collects the validation and control results. Model–harness checkpoint pairings show increasing compatibility loss with checkpoint distance. The measurement sweep separates behavioral change from uncertainty sampling, and the matched-intervention study tracks attribution accuracy over training. Schedule, routing, transfer, horizon, and compute controls provide complementary checks on the allocation rule. The following appendix sections give the underlying measurements and protocols.

Table 2: Results of the validation and control studies on Terminal-Bench 2 at round 40, with three seeds unless noted.

Study	Result	Details
Checkpoint compatibility	Off-diagonal model×harness pairs lose 5.2/10.0/13.6 points at checkpoint distance 1/2/3.	App. O
Change and uncertainty	At $N = 8$ the raw magnitude correlates $0.26 \to 0.44$ with $\bar{p}(1-\bar{p})$ over the run. The null-corrected score correlates $0.10 \to 0.15$ and scores 44.6, compared with 40.0 for entropy sampling.	App. L
Measurement budget	Budget-matched scores are 42.6/44.6/44.9/44.1 at $N = 4/8/16/32$. Top-K overlap rises from 0.52 to 0.88.	App. L
Adaptive weighting	A cosine ramp scores 42.9, oracle replay 43.8, and adaptive weighting 44.6 (n.s. at task level). On Qwen3.5-9B, replay scores 19.1 and adaptive weighting 23.4.	App. K
Attribution calibration	Accuracy against matched interventions rises from 58% to 74%, and Youden's index from 0.13 to 0.47. Routed q_t exceeds the random-set rate by 7 points.	App. N
Judge controls	Frozen round-0 judge 66% / 44.0, frozen 122B-A10B 79% / 45.3, and frozen 9B 57% / 31.8.	App. N
Regression routing	Same-side repair scores 43.9, sign-split 43.1, and revert-on-regression with counterpart routing 45.0.	App. M
Factorial transfer	The in-domain interaction is +6.3, about 30% of the gain. Within-domain, Terminal-Bench 2 to SWE-bench Pro, the model and harness contribute +2.2 and +1.1. Across domains the model contributes +0.9 to +1.6, the harness +0.5 to +0.6, and the interaction +0.4 to +0.5.	App. R
Extended horizon	The score plateaus near round 60 at 47.3 (80 rounds, two seeds), with drawdown 0.4.	App. S
Compute matching	Self-Judgment uses 1.0% of input tokens. Raising B6's cap by 9% changes its score from 40.0 to 40.6. Total GPU-hours are within 11% across arms.	App. F

C Analysis of the Allocation Rule

This appendix characterises allocation at a fixed round. Shared model parameters and executable harness edits couple task outcomes across tasks, so long-horizon performance is measured empirically.

C.1 An Exact Mixture Interpretation

Fix receiver a and write $d_x = \Delta^{\bar{a}}(x) \geq 0$, $z_x = \mathbb{1}[b(x) = a]$, $D = \sum_x d_x$, and $M = \sum_x z_x$, with sums over the current candidate panel.

Proposition 1 (Allocation mixture)**.** *If $D > 0$ and $M > 0$, the model sampling distribution has the exact form*

$$P_\beta(x) = (1 - \alpha_\beta) P_\Delta(x) + \alpha_\beta P_b(x), \qquad \alpha_\beta = \frac{\beta M}{D + \beta M}, \tag{8}$$

where $P_\Delta(x) = d_x/D$ and $P_b(x) = z_x/M$.

Proof. Substituting both distributions into Eq. 8 gives $(d_x + \beta z_x)/(D + \beta M)$, the normalised score of Eq. 4. □

If $M = 0$ the attribution term contributes no mass. If $D = 0$ and $\beta M > 0$ sampling is uniform within the attributed group. If both vanish the implementation samples uniformly over the panel. The attribution share α_β depends on the number of matching labels as well as on the coefficient, so the routing-mass handoff reported in Appendix J.2 can arise through the relative scales of the

two signals even at fixed judgment quality, and class prevalence enters directly through M. For any task-level statistic h_x,

$$\mathbb{E}_{P_\beta}[h] - \mathbb{E}_{P_\Delta}[h] = \alpha_\beta\big(\mathbb{E}_{P_b}[h] - \mathbb{E}_{P_\Delta}[h]\big). \tag{9}$$

Attribution improves the expected statistic exactly when its matching group has the larger group mean. The success rate of executed interventions on routed tasks does not estimate the alternative group's mean. This identity is what motivates the matched-intervention calibration set of Appendix N, which measures both interventions on the same randomly chosen tasks.

C.2 Coefficient and Ranking Sensitivity

Proposition 2 (Sensitivity to feedback)**.** *For fixed d and z with $D > 0$ and $M > 0$, two nonnegative coefficients satisfy*

$$\mathrm{TV}(P_\beta, P_{\beta'}) = |\alpha_\beta - \alpha_{\beta'}|\,\mathrm{TV}(P_b, P_\Delta) \le \frac{DM|\beta - \beta'|}{(D + \beta M)(D + \beta' M)}, \tag{10}$$

and the implemented coefficient satisfies $|\beta(q) - \beta(q')| \le 2\beta_{\max}|q - q'|$.

Proof. Subtract the two mixtures and use the definition of total variation. Subtracting the rational expressions for α gives the bound. The positive-part function is 1-Lipschitz. □

Proposition 3 (Top-K stability)**.** *Let w and $\widetilde{w}$ be two score vectors with $\|w - \widetilde{w}\|_\infty \le e$. If the gap between the Kth and $(K+1)$th largest entries of w exceeds $2e$, their Top-K sets agree. For fixed labels, $\|w - \widetilde{w}\|_\infty \le \|\Delta - \widetilde{\Delta}\|_\infty + 2\beta_{\max}|q - \widetilde{q}|$.*

Proof. Every selected score exceeds every excluded score by more than $2e$, so perturbing each by at most e preserves their order across the boundary. The second claim follows from the score definition and Proposition 2. □

Top-K margins and resampling overlap are direct diagnostics of the harness selector. Appendix L reports the overlap of Top-K sets between independent measurement draws as a function of N.

C.3 Resolution of Behavioral Change and the Null-Corrected Score

Proposition 4 (Success and conditional-cost uncertainty)**.** *For N independent rollouts per state, let $\hat{D} = \hat{p}_+ - \hat{p}_-$ and $D = p_+ - p_-$. For a panel of m tasks, with probability at least $1 - \zeta$, $\max_x |\hat{D}(x) - D(x)| \le \sqrt{\log(2m/\zeta)/N}$. For successful costs in $[0, C]$, conditional on at least $s > 0$ successes in each state and task, the maximum error of the $2m$ successful-cost means is at most $e_c = C\sqrt{\log(4m/\zeta)/(2s)}$ with conditional probability at least $1 - \zeta$.*

Proof. Changing one Bernoulli outcome changes $\hat{D}$ by at most $1/N$; McDiarmid's inequality and a union bound give the first claim. Conditional on the success counts, successful costs are independent draws from the conditional cost distribution, and Hoeffding's inequality with a union bound gives the second. □

With the fixed reference scale $c_{\mathrm{ref}}(x)$ used in Eq. 5, the cost term's error is at most $2\lambda\rho_c e_c / c_{\mathrm{ref}}(x)$, and the reliability factor $\rho_c = \min(s_-, s_+)/(\min(s_-, s_+) + 1)$ equals zero when either success count is zero, one half at a single success, and approaches one as the counts grow. The cost term therefore switches on at the first success and is damped while few successes support it.

At $N = 8$ the uniform bound is too loose to certify fine-grained rankings, so we use exact binomial diagnostics. The probability that a truly changing task receives zero success-change score is

$$\Pr(\hat{D} = 0) = \sum_{j=0}^{N} \binom{N}{j}^2 p_+^j p_-^j (1 - p_+)^{N-j} (1 - p_-)^{N-j}, \tag{11}$$

and the probability of a wrong sign sums the two independent binomial probabilities over $j_+ < j_-$. Under no change, the null magnitude

$$e_0(\bar{p}) = \mathbb{E}\big[|\hat{D}| \,\big|\, p_+ = p_- = \bar{p}\big] = \sum_{a,b} \frac{|a-b|}{N} \binom{N}{a} \binom{N}{b} \bar{p}^{\,a+b} (1-\bar{p})^{2N-a-b} \tag{12}$$

is largest at $\bar{p} = 1/2$ (0.273 at $N = 4$, 0.196 at $N = 8$, and 0.140 at $N = 16$) and is the quantity that makes raw $|\hat{D}|$ resemble an uncertainty score. The *null-corrected* change score of Eq. 5 subtracts it from the success component, so that a task whose observed change is no larger than the mean magnitude that sampling alone produces at its success level carries no success-change weight. Clipping at zero keeps the expected corrected success component positive under no change, so the correction lowers the noise contribution and leaves part of it. At $p_- = p_+ = 0.5$ the expected corrected success component is 0.095 at $N = 4$, 0.066 at $N = 8$, and 0.044 at $N = 16$, against 0.273, 0.196, and 0.140 for the raw magnitude. Table 3 lists the diagnostics. Appendix L compares raw and null-corrected routing across N.

Table 3: Exact binomial measurement diagnostics for the success component. The unchanged case has $p_- = p_+ = 0.5$; the changing case has $p_- = 0.4$ and $p_+ = 0.6$.

N	4	8	16	32
no change ($p_- = p_+ = 0.5$): $\Pr(\hat{D} \neq 0)$	72.7%	80.4%	86.0%	90.1%
no change: $e_0 = \mathbb{E}\lvert\hat{D}\rvert$	0.273	0.196	0.140	0.099
no change: $\mathbb{E}\max\{0, \lvert\hat{D}\rvert - e_0(\bar{p})\}$	0.095	0.066	0.044	0.031
change $0.4 \to 0.6$: $\Pr(\hat{D} = 0)$	23.2%	14.2%	7.3%	2.7%
change $0.4 \to 0.6$: $\Pr(\hat{D} < 0)$ (wrong sign)	17.4%	14.2%	9.2%	4.0%

C.4 Relation between the Routing Score and the Reference Utility

Write $\hat{G}(x) = (\hat{p}_+ - \hat{p}_-) - \lambda\rho_c(\hat{c}_+ - \hat{c}_-)/c_{\mathrm{ref}}(x)$ for the intervention gain of Eq. 7. It uses the fixed denominator of the reference utility in Eq. 1 and applies ρ_c to the empirical cost difference. Let $\Delta^{\mathrm{raw}}(x)$ be the score of Eq. 5 without e_0. It satisfies $|\hat{G}(x)| \leq \Delta^{\mathrm{raw}}(x)$ by the triangle inequality, with equality when success and cost effects have the same sign in the feedback gain, and the routing score satisfies $\Delta(x) \leq \Delta^{\mathrm{raw}}(x)$. An update that raises success and cost by offsetting amounts can therefore have $\hat{G} \approx 0$ and $\Delta > 0$. The score tracks behavioral displacement for routing, while $\hat{G} > 0$ denotes an improvement under the reliability-weighted feedback statistic. Removing the shrinkage factor recovers the direct plug-in change in reference utility.

C.5 Prevalence-Aware Reliability and Selection Bias

The feedback statistic q_t is the success rate of executed interventions on routed tasks. Let π be the prevalence of harness-responsible failures, and let TPR and TNR be the judge's true-positive and true-negative rates for the harness label. Accuracy is $\pi\,\mathrm{TPR} + (1-\pi)\mathrm{TNR}$ and depends on π, which the matched-intervention results in Appendix J.2 show moving over rounds. The Youden index $J = \mathrm{TPR} + \mathrm{TNR} - 1$ (Youden, 1950) does not depend on π and equals $2\,\mathrm{acc} - 1$ when the classes are equally prevalent or when the two rates coincide. The calibrated coefficient $\beta_t^{\mathrm{cal}} = \beta_{\max}\max(0, J_{t-1}^{\mathrm{cal}})$ uses the confusion matrix measured on the random calibration set of Appendix N. Because that set is sampled independently of the router, $\Pr(\hat{G} > 0)$ on it is free of the selective-observation effect that inflates q_t on routed tasks. Table 16 reports both and the difference between them.

C.6 The Model's Training Distribution

The model draws task groups from P_t^{model} and uses binary verifier rewards. At a fixed router, its expected group loss is

$$\mathcal{L}_t(\theta) = \mathbb{E}_{x \sim P_t^{\mathrm{model}}} \mathbb{E}_{\tau_{1:N} \sim \pi_{\theta_t}(\cdot|x, H_t)} [\ell_{\mathrm{GRPO}}(\theta; \tau_{1:N})]. \tag{13}$$

Group advantages are centred and normalised within each task using verifier outcomes, and the router is fixed during the optimisation step. Equation 13 specifies an adaptive curriculum without importance correction. The uniform-task functional J remains the reference for system evaluation.

D Protocol Details

For Terminal-Bench 2 the held-out set is the complete Terminal-Bench 2.0 release of 89 tasks (Merrill et al., 2026), and no Terminal-Bench 2.0 task is used for evolution. The evolution pool is a disjoint set of 312 terminal tasks assembled from Terminal-Bench 1.0 tasks that were not carried into 2.0 and from community task sets in the same Harbor format. A candidate is kept when its verifier is deterministic, which is checked by repeated runs of the reference solution in an idle container, and is removed when its name matches a 2.0 task, when the cosine similarity between its description embedding and any 2.0 description is at least 0.90, or when its verifier hash equals a 2.0 verifier hash. Terminal-Bench 2.1 (Terminal-Bench Team, 2026) patched 28 of the 89 tasks, and Appendix G evaluates the round-40 systems on it. For SWE-bench Pro (Jimenez et al., 2024; Deng et al., 2025) evolution trajectories are drawn from the disjoint SWE-Gym pool (Pan et al., 2024), and the full public set of 731 instances from 11 repositories is held out. For GAIA (Mialon et al., 2024), 65 of the 165 validation tasks form the evolution pool and the remaining 100 are held out, using a permutation of the validation task identifiers with sampling seed 17 stratified by difficulty level. The round-40 system of each compared arm, taken from its first seed, is also submitted once to the private 300-task GAIA test set, whose answers are not public, and GAIA answers are scored with the official quasi-exact-match scorer. SearchQA combines HotpotQA (Yang et al., 2018) and 2WikiMultiHopQA (Ho et al., 2020) in equal proportions. Evolution uses 1,000 questions from each development split and held-out evaluation uses a disjoint 500 from each, all drawn with seed 17. Retrieval uses the December 2018 Wikipedia snapshot indexed with BM25 and a dense retriever behind a local endpoint that returns the top five passages, and answers are scored by exact match after lowercasing and removal of articles and punctuation. Held-out tasks are never used to accept edits, tune hyperparameters, select checkpoints, or terminate a run.

Every arm uses the same Python-package runtime and editable surface, and an edit is a source patch to that package. We initialise H_0 with Terminus-2 (Merrill et al., 2026; Laude Institute, 2026) for Terminal-Bench 2, OpenHands (Wang et al., 2025) for SWE-bench Pro, OWL (Hu et al., 2025a) for GAIA, and Search-o1 (Li et al., 2025) for SearchQA. The second and third human harnesses of Table 1 are OpenHands and OpenCode on Terminal-Bench 2, SWE-agent (Yang et al., 2024) and smolagents (Roucher et al., 2025) on SWE-bench Pro, smolagents and a ReAct loop (Yao et al., 2023) on GAIA, and the ReAct loop and smolagents on SearchQA. Task execution runs in non-root containers with the network policy of the task, since most Terminal-Bench and SWE-bench tasks disallow outbound network and search tasks allow the retrieval endpoint only, with no credentials or secrets mounted. A path guard rejects any patch touching the verifier, the regression suite, the task definitions, or the evaluation loop, and a static safety linter rejects patches that add network calls where the task disallows them, escalate shell privileges, write outside the workspace, or read environment secrets. Appendix T reports gate outcomes and harness complexity over rounds.

The co-evolution baselines are HELIX-A (Fan & Huang, 2026), HarnessX (Chen et al., 2026b), HarnessForge (Chen et al., 2026a), HASE (Luo et al., 2026), and SIA (Hebbar et al., 2026), and the harness-only baselines are AHE (Lin et al., 2026) and Self-Harness (Zhang et al., 2026a). We also include three static human harnesses per benchmark and budget-matched test-time scaling through verified parallel sampling and sequential self-refinement. Appendix E documents every modification made to each baseline and reports the baselines under their native protocols.

Let y_t be held-out success averaged over five seeds. Curve metrics are computed on this seed-mean curve, except σ_Δ and the worst single-round drop, which are computed on the curve of each seed and then averaged over seeds. Appendix tables list the default configuration with its five-seed main-run value and run only the varied configurations at the number of seeds stated in each caption. We report final score y_{40}, drawdown from peak $D_{\mathrm{pf}} = \max_t y_t - y_{40} \geq 0$, the least-squares slope over rounds 20–40, $y_{40} - y_{15}$, standard deviation σ_Δ of round-to-round changes, area under the learning curve $\mathrm{AUC}_{1\text{–}40}$, and mean success over the last five rounds. The saturation round t_{95} is the first round at which the smoothed curve reaches 95% of its maximum gain over y_0, where smoothing is a centered three-round moving average whose window shrinks to two rounds at rounds 0 and

40. A run collapses when its smoothed curve falls at least 3 points below its peak and does not recover, and the collapse round is the first round after that peak. These summaries describe different aspects of one curve. Shaded curves are 95% confidence intervals of the seed mean at a fixed task split. For main comparisons, seed outcomes are averaged within each task before resampling and differences are formed per task. A percentile bootstrap with 10,000 resamples of tasks provides task-population intervals (Efron et al., 2000). On SWE-bench Pro the resampling unit is the repository, so tasks from the same repository stay together and the eleven public repositories are the clusters. A sign-flip permutation test on the task-level mean differences provides p-values, with 10,000 random sign patterns and exact enumeration of the 2^{11} patterns at the repository level on SWE-bench Pro (Table 8). The private GAIA test reports aggregate scores only, so its interval and test are unpaired.

Each arm has an evolution-rollout cap of $B = 16{,}384$ per round. The candidate panel is drawn uniformly from the evolution pool, before scores are computed, and both receivers rank or sample from it. It contains 44 tasks on Terminal-Bench 2, 184 on SWE-bench Pro and SearchQA, and the whole pool of 65 tasks on GAIA. On the first three benchmarks a quarter of the panel is replaced each round. A round measures two behavioral changes whose comparisons share the endpoint (θ_t, H_t), and the state (θ_t, H_{t-1}) was measured at the end of the previous round for every task that stays in the panel. Tasks that stay therefore need two fresh states, (θ_t, H_t) and (θ_{t+1}, H_t), and tasks new to the panel need a third, so a round executes $2.25\,Nm$ measurement rollouts on average with $N = 8$ (and $2Nm$ on GAIA). The matched-intervention calibration step (Appendix N) costs $16 \times (16+4+4) = 384$ rollouts per round, because its starting-state success is taken from the step-(iii) measurement and its discarded harness branches are gated without the regression suite. Edit validation against the 32-task regression suite costs 128 rollouts per proposal that reaches that stage, one per round on average. The remaining allowance determines the realized number of GRPO iterations of 128 tasks $\times$ 8 rollouts, and Table 4 lists the realized counts, which are 12 to 14 and fall below the nominal limit of 16. Every execution is charged once. A trajectory is reused only under an identical model, harness, task, and sampling-state fingerprint, and measurement rollouts of a task that is also sampled into a training group are pooled with that group's rollouts under the same state, which is why some case-study estimates in Appendix U have resolution finer than $1/8$. Reference costs and the round-0 evaluation are measured once before round 1 and are not charged to any round. Held-out evaluation is counted separately and never charged. In the k sweep, the training rollouts of a round stay at the default allocation of 1,792 groups of eight on Terminal-Bench 2 and an iteration draws $1{,}792/k$ tasks, so measurement, calibration, and validation charges are identical for every k.

Table 4: Composition of the per-round rollout cap and the realized number of GRPO iterations.

Per-round rollouts (cap $B = 16{,}384$)	Terminal-Bench 2	SWE-bench Pro	GAIA	SearchQA
candidate panel m	44	184	65	184
share of the panel replaced each round	25%	25%	none	25%
fresh measurement states per task, average	2.25	2.25	2	2.25
measurement rollouts, $N = 8$	792	3,312	1,040	3,312
calibration, 16 tasks $\times$ (16 training + 4 + 4 evaluation)	384	384	384	384
edit validation, 32 tasks $\times$ 4 rollouts, one evaluation	128	128	128	128
training rollouts available	15,080	12,560	14,832	12,560
realized GRPO iterations (1,024 rollouts each)	14	12	14	12
unused slots	744	272	496	272
held-out evaluation (reported separately)	356	2,924	400	4,000

Unless noted, Qwen3.5-35B-A3B is the task model, the judge, and the evolver. Appendix P lists the checkpoints of every backbone in Table 20. The model update trains rank-64 LoRA adapters on the attention and dense or shared-expert MLP projections with the base weights and routed experts frozen, and the main configuration uses 8 NVIDIA H100 80GB GPUs.

GRPO (Shao et al., 2024) with group size 8, 128 tasks per iteration drawn from P_t^{model}, learning rate 2×10^{-5} for the adapters, clip 0.2, no KL penalty to the initial policy, and sequence-level advantage normalisation within each group. Groups with zero outcome variance are dropped before the gradient step and counted in the budget (Zhai & Wang, 2026). We optimise with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, maximum gradient norm 1.0) in bfloat16 with DeepSpeed ZeRO-3 sharding of the frozen weights, a cosine learning-rate schedule with 3% warm-up and no restart between rounds, a maximum context of 32,768 tokens, and a cap of 8,192 newly generated tokens per model turn. The trajectory cap is 120,000 generated tokens, and $c(\tau)$ sums generated

tokens over turns. Trajectories exceeding the context are truncated from the oldest non-system interaction while retaining the task specification and system context. Rollouts use temperature 0.8 and top-$p = 0.95$, and each model update starts from the parameters of the preceding round. Rollout generation is served asynchronously by vLLM and training runs on the same GPU pool between rollout phases, with gradient checkpointing and FlashAttention-2 enabled throughout.

The evolver receives $K = 16$ tasks selected as $\mathrm{TopK}(w_{\mathrm{harness}})$, their full trajectories, and the current harness source. It generates with temperature 0.6, top-$p = 0.95$, and at most $32{,}768$ output tokens, makes one proposal per step, and may retry once when the path guard, linter, or compilation rejects the proposal. It may edit system instructions, tool implementations and registrations, context and memory management, and control flow and sub-agent delegation. An edit is committed only if it passes the path guard and safety linter, compiles, passes the harness's own unit tests, and does not regress the fixed 32-task regression suite drawn from the evolution pool. The regression suite never contains held-out tasks. Every proposed and committed diff is logged, and Appendix T reports a leakage scan of all diffs for task identifiers, expected-answer literals, and benchmark-specific paths.

Self-Judgment is one additional call per failed rollout at the freshly measured states of a round, (θ_t, H_t) in step (i) and (θ_{t+1}, H_t) in step (iii), after the verifier returns 0. The call receives the task statement, the complete trajectory including every tool call and its output, and the harness tool manifest as names with one-line descriptions, and it does not receive the harness source. The response is one token from $\{\text{HARNESS}, \text{MODEL}\}$ followed by a free-text reason that is logged and never used for routing, with greedy decoding and no retry. The calls are excluded from reinforcement learning, are never training targets, and are never appended to the trajectory that produced them. Per-task aggregation is a majority vote over failed trajectories with ties broken toward MODEL, because a harness edit changes every later trajectory while a model update trained on one task's batch does not. Appendix N reports the four-way variant with BOTH and UNSURE and the effect of the tie rule. Tasks whose rollouts all succeed receive $b(x) = \varnothing$ and no attribution weight. The model allocation of step (ii) uses the labels aggregated from step (i), and the harness allocation of step (iv) uses the labels aggregated from step (iii), so each receiver sees judgments made at the state on which it acts.

The two states of each comparison are measured on the same panel with $N = 8$ rollouts each at the training temperature, so the success estimates share the panel and rollout draws are independent conditional on each state. The states (θ_t, H_t) and (θ_{t+1}, H_t) are measured in the current round at steps (i) and (iii). The state (θ_t, H_{t-1}) is the state measured at step (iii) of the preceding round when the task was in that panel and is measured at step (i) otherwise. The cost term uses mean generated tokens over successful rollouts, normalised by the fixed reference $c_{\mathrm{ref}}(x)$ of the initial system, and is shrunk by the reliability factor ρ_c of Eq. 5. When either state has no successful rollout, $\rho_c = 0$ and Δ reduces to the corrected success term. Appendix L sweeps N and compares the null-corrected and raw scores.

The feedback statistic q_t counts routed task occurrences. An occurrence is routed when a task enters the model batch with the label MODEL or enters the harness Top-K with the label HARNESS. Its outcome is whether $\hat{G} > 0$ for the change measured across the corresponding update. A model-side outcome becomes observable at step (iii) of the same round, where the panel measurement compares the states before and after the model update. A harness-side outcome becomes observable at step (i) of the following round when the task stays in the panel, and a harness-side occurrence whose task rotates out of the panel is dropped from the statistic. Panel rotation is drawn before scores are computed, so the dropped occurrences form a random subset of the routed ones, and the statistic never replays historical checkpoints. An outcome is assigned to the round in which it is observed, model-side and harness-side outcomes count with equal weight per occurrence, q_t is the fraction of positive outcomes observed in rounds $t-2$ to t, and the previous value is kept when that window is empty. At round 1 no harness update has occurred, so $\Delta_0^{\mathrm{H}} = 0$ and $q_0 = 1/2$ gives $\beta_1 = 0$, the model samples uniformly from the panel, and the harness ranks tasks by Δ_1^{M} alone. Ties in Top-K are broken by the ascending hash of the task identifier, and slots that remain when fewer than K tasks have positive weight are filled uniformly from the panel. When fewer than 16 failed panel tasks are available for calibration, all of them are used.

We set $\lambda = 0.2$, $\beta_{\max} = 0.5$, $K = 16$, and $N = 8$, fixed from a 20-round pilot on the Terminal-Bench evolution pool before the main sweep and unchanged across benchmarks and arms. Appendix V reports sensitivity to λ and K, where the Terminal-Bench 2 panel is enlarged to $2K$ tasks at $K = 64$, since K would otherwise exceed the 44-task panel.

One round consists of (i) selecting the candidate panel and measuring $\Delta^{\mathrm{H}}_{t-1}$ on it, (ii) the realized number of policy-update iterations, (iii) measuring Δ^{M}_{t}, (iv) one harness-evolution step, (v) the matched-intervention calibration step on 16 failed panel tasks at the state of step (iii), and (vi) a held-out evaluation used only for reporting. *Self-Judgment* labels come from the failed rollouts at the fresh states of step (i), which the model allocation uses, and of step (iii), which the harness allocation and the calibration step use. The coefficient β is updated from the routed interventions observed in the round. Algorithm 1 states the recursion.

Algorithm 1 One round of CoEvo-RSI

Require: Current model θ_t, harness H_t, previous harness H_{t-1}, coefficient β_t

1: Measure harness-induced change $\Delta^{\mathrm{H}}_{t-1}$ from $(\theta_t, H_{t-1}) \to (\theta_t, H_t)$ (one fresh state, plus one for tasks new to the panel) and obtain *Self-Judgment* labels $b^{(1)}_t$ from the failed rollouts at (θ_t, H_t)

2: $w_{\mathrm{model},t}(x) \leftarrow \Delta^{\mathrm{H}}_{t-1}(x) + \beta_t \mathbb{1}[b^{(1)}_t(x) = \mathrm{model}]$

3: $\theta_{t+1} \leftarrow \mathrm{RL}\left(\theta_t \mid H_t, \mathrm{Normalize}(w_{\mathrm{model},t})\right)$

4: Measure model-induced change Δ^{M}_{t} from $(\theta_t, H_t) \to (\theta_{t+1}, H_t)$ (one fresh state) and obtain labels $b^{(2)}_t$ from the failed rollouts at (θ_{t+1}, H_t)

5: $w_{\mathrm{harness},t}(x) \leftarrow \Delta^{\mathrm{M}}_{t}(x) + \beta_t \mathbb{1}[b^{(2)}_t(x) = \mathrm{harness}]$

6: $H_{t+1} \leftarrow \mathcal{E}\left(H_t \mid \theta_{t+1}, \mathrm{TopK}(w_{\mathrm{harness},t})\right)$, subject to the acceptance gate

7: Run matched model-only and harness-only interventions on 16 failed panel tasks from the state (θ_{t+1}, H_t), and record $\ell(x)$ and J^{cal}_t against $b^{(2)}_t$

8: Update q_t from the routed interventions observed in this round, set $\beta_{t+1} = \beta_{\max} \max(0, 2q_t - 1)$, and advance $t \leftarrow t + 1$

E Baseline Protocols and Change Manifest

Table 1 compares adaptations that share a runtime, evolver backbone, optimiser, rollout cap, and held-out split. Table 5 records the published component and its implementation in each adaptation. Table 6 reports native-protocol runs under the same benchmarks and rollout cap, making the effect of standardization visible.

Table 6 runs each baseline with its own update rule, gate, optimiser, and schedule where these are specified by the publication or its code, changing only what is required to run on our benchmarks and under the same rollout cap, namely the task pools, the held-out split, and the backbone. For HarnessForge the native run uses SFT alignment. For HarnessX the AEGIS roles are instantiated with the published prompts. For SIA the feedback agent and optimiser follow the reference implementation. For HELIX we run one native build–update–rebuild round with a 65-candidate portfolio (which alone consumes about three rounds of our cap) and then freeze the system, and separately iterate the cycle under the cap (HELIX-A). Three seeds per cell.

Table 6 shows similar rankings under native and standardized protocols. On Terminal-Bench 2, standardized scores are within 1.2 points of native scores for each co-evolution system except SIA, for which the shared GRPO optimiser adds 2.3 points. On Terminal-Bench 2 the two harness-only systems score slightly higher under their native gates, and on GAIA Self-Harness does while AHE scores 0.7 points lower. CoEvo-RSI's margin over the strongest baseline is $+5.0$ points against the native HELIX-A run and $+4.5$ against the standardized adaptation. Table 1 therefore compares standardized systems that share the runtime, evolver backbone, optimiser, cap, and split and keep the update schedules, gates, and candidate generation listed in Table 5. The controlled comparison of allocation rules is the B0–B7 study of §3.4, and Table 6 shows the behavior of the systems under the available native protocols.

The native HELIX run uses its 65-candidate portfolio for one cycle, which consumes about three rounds of the shared cap. HELIX-A uses eight candidates per round across 40 rounds. HarnessX's harness-primitive library is mapped to the editable surface, and AHE's observability tooling is implemented against the shared trace format.

Table 5: Change manifest for the standardized adaptations. "Native component" names the mechanism as published. "Standardized adaptation" states what runs in our comparison.

System	native component	standardized adaptation	reason
HarnessForge	fault-guided harness tailoring; harness-conditioned SFT alignment; evaluation-set gate	tailoring kept as the evolver prompt; SFT replaced by GRPO on the same trajectories (SFT variant reported); gate moved to an evolution-validation slice	shared optimizer; no held-out leakage
HASE	unified action space (solve or edit within a trajectory); PoC gate on measured gain	action space kept as an extra tool (Appendix H); gate kept; its evaluation set replaced by the validation slice	no held-out leakage
HarnessX	typed primitive substitution; AEGIS multi-agent evolution; FIFO trajectory buffer; training-signal export	substitution algebra kept over our editable surface; AEGIS roles run by the shared evolver backbone; FIFO buffer kept; export kept	shared evolver backbone
SIA	feedback agent chooses harness or weight update; own optimizer	feedback agent kept (same backbone); optimizer replaced by GRPO	shared optimizer
HELIX	source-traceable substrate; harness portfolio (65 candidates); verified records from a sibling slice; one build–update–rebuild round	substrate mapped to our editable package; portfolio of 8 candidates per round under the rollout cap; verified records kept; cycle iterated for 40 rounds	budget cap; horizon
AHE	three observability pillars; decision observability on edits; frozen weights	pillars kept as evolver context; frozen weights kept	harness-only comparator
Self-Harness	weakness mining; proposal generation; dual regression gate; frozen weights	all kept; gate slice replaced by the validation slice	harness-only comparator

Table 6: Native versus standardized protocols at round 40 (HELIX native: after its single round), three seeds, mean ± standard deviation. †Harness-only systems.

		Terminal-Bench 2		GAIA	
System	native protocol elements retained	native	standardized	native	standardized
HarnessForge	fault-guided tailoring, SFT alignment, own schedule	36.9 ±1.3	37.6 ±1.1	34.4 ±1.5	35.6 ±1.2
HASE	unified action space, PoC gate, own evolver prompt	37.8 ±1.2	38.4 ±1.0	36.5 ±1.4	37.1 ±1.3
HarnessX	primitive substitution, AEGIS multi-agent evolution	38.1 ±1.6	39.3 ±1.2	25.9 ±2.1	26.7 ±1.9
SIA	feedback agent, weight and harness updates	27.4 ±2.0	29.7 ±1.8	26.6 ±2.2	27.9 ±2.0
AHE†	three observability pillars, own gate	35.3 ±1.2	34.9 ±1.1	30.1 ±1.4	30.8 ±1.3
Self-Harness†	weakness mining, dual gate	29.6 ±1.9	28.9 ±1.7	25.0 ±2.0	24.3 ±1.9
HELIX (one native round, then frozen)	portfolio build, verified records, one rebuild	31.5 ±1.0	—	30.2 ±1.2	—
HELIX-A (iterated for 40 rounds)	portfolio build–update–rebuild each round	39.6 ±1.3	40.1 ±1.1	35.7 ±1.5	36.2 ±1.4
CoEvo-RSI	—	—	44.6 ±0.9	—	43.5 ±1.2

The HarnessX buffer holds the 256 most recent trajectories, and its evolver receives K of them by recency. The HarnessForge tailoring prompt receives the K failed tasks with the lowest success rate. The HASE proof-of-concept gate accepts an edit when it raises success on at least one of 32 gate tasks and lowers none. The HELIX-A portfolio holds eight candidates per round, four generated from fault clusters of the routed tasks, two recombining the two best candidates of the previous round, and two perturbing the incumbent. Each candidate is scored on the 32-task regression suite with two rollouts, and the best candidate that does not regress the incumbent is committed while its verified trajectories join the model's training pool. In the allocation study, B1 draws all trajectories to both components, B0 draws the evolver's K tasks independently and uniformly from the panel, and B2 samples model tasks uniformly from the panel tasks with at least one success while keeping all eight rollouts of each in its group, so groups retain outcome variance whenever a rollout fails. The SIA feedback agent uses the prompt of Appendix H. All evolvers generate with temperature 0.6, top-$p = 0.95$, and at most 32,768 output tokens, with one proposal per step and one retry after a rejection by a cheap gate.

F Compute Ledger and Budget Matching

Equal environment-rollout caps are one component of compute matching. CoEvo-RSI issues *Self-Judgment* calls on failed panel measurements and runs the matched-intervention calibration step,

the baselines' evolvers and feedback agents incur inference of their own, and the arms differ in trajectory length, which changes generated tokens at a fixed rollout count. Table 7 reports, for one 40-round run on Terminal-Bench 2, the executed environment rollouts by category and their sum against the cap of 655,360, generated and input tokens, the number and token cost of *Self-Judgment* and evolver calls, task-model inference and training GPU-hours, and wall-clock time on the 8×H100 configuration.

Table 7: Execution ledger for one 40-round run on Terminal-Bench 2 with Qwen3.5-35B-A3B. Rollout counts are executions charged to the cap except the held-out row, and token counts are billions. Rollout generation and training alternate on the same eight GPUs, so wall-clock time is total GPU-hours divided by 192 GPU-hours per day.

Per 40-round run, Terminal-Bench 2	CoEvo-RSI	HELIX-A	HASE	B6 entropy	B1 all-to-both
executions, training (14 or 15 iterations per round)	573,440	614,400	614,400	614,400	614,400
executions, measurement	31,680	0	0	14,080	0
executions, calibration	15,360	0	0	0	0
executions, validation and gate evaluations	5,120	20,480	24,320	5,120	5,120
charged executions, total	625,600	634,880	638,720	633,600	619,520
cap of $40 \times 16{,}384$ minus charged executions	29,760	20,480	16,640	21,760	35,840
executions, held-out (not charged)	14,240	14,240	14,240	14,240	14,240
generated tokens (B)	5.9	6.2	6.2	6.3	6.6
input tokens (B)	39.9	40.9	40.6	40.6	43.8
Self-Judgment calls / input tokens (B)	20.6k / 0.4	0 / 0	0 / 0	0 / 0	0 / 0
evolver calls / tokens (B)	680 / 0.19	320 / 0.31	1,210 / 0.24	40 / 0.05	40 / 0.11
task-model inference GPU-h (H100)	1,870	1,910	1,890	1,880	2,050
training GPU-h	570	630	620	620	640
total GPU-h	2,440	2,540	2,510	2,500	2,690
wall-clock days on 8×H100 (total GPU-h / 192)	12.7	13.2	13.1	13.0	14.0

CoEvo-RSI is charged for its measurement states, 2.25 per panel task on average with $N = 8$ rollouts each, for the calibration step, and for one regression evaluation per round. HELIX-A is charged 512 rollouts per round for evaluating its eight candidate harnesses on the 32-task regression suite with two rollouts each. HASE is charged 608 rollouts per round for its proof-of-concept gate evaluations. B6 is charged one fresh measurement state of $N = 8$ rollouts per round on the panel to compute the entropy score, and the same regression evaluation. B0 and B1 are charged for the regression evaluation only. The baselines realize 15 GRPO iterations per round on Terminal-Bench 2 and CoEvo-RSI realizes 14, and the arms differ in the rollouts left unused. Reference costs and the round-0 evaluation are measured once before round 1 and are not charged to any round.

CoEvo-RSI's additional inference is small relative to the rollouts themselves. *Self-Judgment* calls consume 0.4B input tokens, about 1.0% of its input tokens, and measurement and calibration together take 7.2% of the rollout cap. Its task-model inference GPU-hours are within 3% of HELIX-A and HASE and below the all-to-both control, whose trajectories are longer. Its training GPU-hours are 8% to 11% below those of the other arms because it realizes 14 iterations per round against 15, and total GPU-hours are within 11%.

We re-ran B6 (entropy sampling) and B0 (uniform) with their rollout cap raised by 9%, which exceeds CoEvo-RSI's combined measurement, calibration, and *Self-Judgment* overhead of about 8%. Generated tokens follow trajectory length, which differs across arms, so this control equalizes the rollout allowance and lets token totals differ. The enlarged runs execute more rollouts than the default runs, and the default B6 run already generates 6.3B tokens against 5.9B for CoEvo-RSI (Table 7). B6 moves from 40.0 to 40.6 and B0 from 35.3 to 35.7 on Terminal-Bench 2 (three seeds). The ordering is unchanged and the margin to CoEvo-RSI remains at least 4.0 points.

The main design has four benchmarks, nine evolving arms, and five seeds, giving 180 runs, and the appendix studies add 214 runs at three seeds on Terminal-Bench 2 and GAIA, with two seeds for the λ/K sweep and the 80-round extension. Summing the per-run ledgers, the main design corresponds to about 4.5×10^5 H100 GPU-hours and the appendix studies to about 6.2×10^5, of which about 3.3×10^5 belong to the scale sweep because its three largest backbones use 16 to 64 GPUs per run.

Table 8: Paired task-level comparisons at round 40. Seed outcomes are averaged within each task before resampling, and the mean difference is over tasks. The bootstrap interval resamples tasks, or repositories as clusters on SWE-bench Pro, with 10,000 draws. The seed interval is the 95% interval of the seed mean. The permutation test flips signs of task-level mean differences, at the repository level on SWE-bench Pro. The GAIA private test provides aggregate scores only, so its interval is an unpaired normal-approximation interval and its test an unpaired two-proportion z test.

Benchmark	Comparison	tasks (clusters)	mean diff.	task-bootstrap 95% CI	seed 95% CI	perm. p
Terminal-Bench 2	ours vs. HELIX-A	89	$+4.5$	$[1.6, 7.3]$	$[3.1, 5.9]$	0.004
Terminal-Bench 2	ours vs. B6 (entropy)	89	$+4.6$	$[1.5, 7.6]$	$[3.0, 6.2]$	0.006
Terminal-Bench 2	ours vs. $\beta = \bar{\beta}$	89	$+4.2$	$[1.0, 7.2]$	$[2.5, 5.9]$	0.010
Terminal-Bench 2	ours vs. oracle-replay schedule	89	$+0.8$	$[-2.1, 3.6]$	$[-0.4, 2.0]$	0.57
SWE-bench Pro	ours vs. HELIX-A	731 (11 repos)	$+3.1$	$[0.9, 5.3]$	$[2.2, 4.0]$	0.006
GAIA	ours vs. HASE	100	$+6.4$	$[2.1, 10.6]$	$[4.7, 8.1]$	0.003
GAIA (private test)	ours vs. HASE, unpaired	300	$+5.3$	$[-2.3, 13.0]$	—	0.17
SearchQA	ours vs. HELIX-A	1,000	$+3.3$	$[2.0, 4.6]$	$[2.6, 4.0]$	< 0.001

G Task-Level Statistics and Hidden-Set Evaluation

Table 8 gives the task-population uncertainty that the seed intervals in the figures do not. The task-bootstrap intervals are two to three times wider than the seed intervals on the small held-out sets. The SWE-bench Pro public split has 11 repositories, which form the clusters, and its interval widens accordingly. The comparison against the oracle-replay β schedule (Appendix K) is not distinguishable from zero at the task level. The four main comparisons remain distinguishable from zero on every benchmark under both intervals.

Table 9: Evaluation on sets never seen during evolution or reporting. GAIA private test scores come from a single submission per system, taken from its first seed, and are integer counts of correct answers out of 300 reported to one decimal. Terminal-Bench 2.1 re-evaluates the frozen round-40 systems on the patched release.

Evaluation set (never used during evolution)	H_0	HarnessX	HASE	HELIX-A	CoEvo-RSI
GAIA private test (300 tasks, leaderboard scoring)	24.3	23.7	32.7	32.0	**38.0**
GAIA validation held-out (100 tasks, for reference)	27.0	26.7	37.1	36.2	**43.5**
Terminal-Bench 2.1 (89 tasks, 28 patched; round-40 systems re-evaluated)	25.1	40.0	39.6	41.2	**45.9**
Terminal-Bench 2.0 (89 tasks, pinned; main results)	23.8	39.3	38.4	40.1	**44.6**

Because the GAIA held-out split shares the validation distribution with the evolution split, each round-40 system is also submitted once to the private test set. Absolute scores are lower on the test set for every system, as is typical for GAIA, and the ordering is preserved with a margin of $+5.3$ points between CoEvo-RSI and HASE. With aggregate scores on 300 tasks the unpaired 95% interval of that margin is $[-2.3, 13.0]$, so the private test shows the same observed ordering and does not by itself separate the two systems. On Terminal-Bench 2.1, which patched 28 tasks, every system scores slightly higher and the ordering is unchanged. Neither hidden set was used during evolution, hyperparameter selection, or checkpoint selection.

H Prompts

We include the four main prompts used in CoEvo-RSI, which are the task agent solving a task, the *Self-Judgment* call on each failed trajectory, the evolver proposing a patch, and, for the baselines that need one, an arbitration or selection call. The four-way *Self-Judgment* variant of Appendix N replaces the two-word answer set with `harness`, `model`, `both`, or `unsure`. The prompts for the remaining benchmarks follow the same template.

H.1 Task agent

The task-agent system prompt is part of the harness and therefore editable, the text below is the round-0 version for Terminal-Bench 2, and braces mark values interpolated from harness configuration at call time.

```
You are operating a Linux shell to complete the task below.  You have a
budget of {max_steps} steps; one step is one tool call and its result.

Work in this order:
  1. Reproduce the stated problem before changing anything.
  2. Read enough of the relevant files to locate the cause.
  3. Make the smallest change that addresses the cause.
  4. Verify by re-running the reproduction from step 1.

Tool results are truncated to {MAX_CAPTURE_LINES} lines.  If a result is
truncated, narrow the command before running it again.

Do not modify anything under tests/.  Do not install packages unless the
task states that the network is available.

TOOLS
{tool_manifest}

TASK
{task_statement}
```

The step limit is 30 on Terminal-Bench 2, 40 on SWE-bench Pro, 20 on GAIA, and 12 on SearchQA. SWE-bench Pro uses the OpenHands system prompt unmodified as its round-0 text. The GAIA and SearchQA prompts follow.

```
You are a research assistant answering the question below with the tools
provided.  You have a budget of {max_steps} steps.

Work in this order:
  1. Break the question into the facts it depends on.
  2. Retrieve or read the sources for each fact, including attached files.
  3. Combine the facts and check any arithmetic.
  4. Give the final answer in the requested format, with no explanation.

TOOLS
{tool_manifest}

QUESTION
{task_statement}
```

```
You answer multi-hop questions by searching a Wikipedia index.  You have a
budget of {max_steps} steps.

Work in this order:
  1. Split the question into sub-questions.
  2. Search for each sub-question and extract the answer span.
  3. Verify each span against a second retrieval.
  4. Answer with the shortest span that answers the question.

TOOLS
{tool_manifest}

QUESTION
{task_statement}
```

H.2 SELF-JUDGMENT

The *Self-Judgment* call is issued once per failed trajectory, after the verifier has returned 0. The call receives the task, the trajectory, and the tool manifest as names and one-line descriptions. Decoding is greedy and there is no retry.

```
You attempted the task below and did not succeed.  The full record of the
attempt follows, with the tools that were available to you.
```

```
Decide which single change is more likely to make this task succeed:

  harness -- the execution environment should change: a tool you needed
             is missing, a tool's output was unusable, information you
             needed was not reachable, or the control flow prevented a
             step you needed to take.
  model   -- the environment was sufficient: the tools you needed existed
             and behaved correctly, and the failure was in how you used
             them.

Answer with exactly one word, `harness` or `model`, then one sentence
giving your reason.  The reason is recorded for analysis and is not
acted on.

Your answer does not affect your score on this task and is not used to
train you.  Answer as accurately as you can, without favouring yourself.

TASK
{task_statement}

TOOLS AVAILABLE
{tool_manifest}

YOUR ATTEMPT
{trajectory}

VERIFIER
failed: {verifier_message}

Answer:
```

Responses are parsed by matching the first occurrence of `harness` or `model` at the start of the response. A response matching neither, or both before either is resolved, is discarded and that trajectory contributes no judgment, this occurs on under 1% of calls in the pilot.

H.3 HARNESS EVOLVER

The harness evolver is called once per round with the $K = 16$ tasks selected by $\mathrm{TopK}(w_{\mathrm{harness}})$. The anti-specialisation guidance matters. The evolver sees 16 of several hundred evolution tasks, and in a pilot without it the highest-scoring patch was frequently one that special-cases a task name or an expected answer, which the regression gate does not catch because such a patch regresses nothing. The leakage scan of Appendix T rejects such patches automatically, and the guidance reduces how often they are proposed.

```
You maintain the execution harness for an LLM agent.  Below are {K} tasks
where the agent recently failed or where its behaviour changed most, each
with the full record of one or more attempts, followed by the current
harness source.

Diagnose what in the harness caused or permitted these outcomes, and make
the smallest set of source changes that would let the agent do better.

You may change:
  * the system instructions given to the agent
  * tool implementations, their signatures, and their registration
  * context and memory management
  * control flow and sub-agent delegation
  * validation and retry logic

You may not change: the regression suite, the verifier, the task
definitions, or the evaluation loop.  A patch touching those paths is
rejected automatically.
```

```
Guidance:
  * Prefer a change that generalises over one that special-cases these
    tasks.  You are seeing {K} of several hundred; a fix keyed to a task
    name, a file path, or an expected answer will be rejected on review.
  * A tool whose output the agent could not act on is a defect even when
    the tool ran without error.
  * If a mechanism added in an earlier round no longer appears in
    successful trajectories, consider removing or narrowing it.
    Removals are edits.
  * If no harness change would help these tasks, emit an empty patch and
    say why.

Emit a unified diff against the source below, then one paragraph of
rationale.  Your patch is committed only if it applies cleanly, compiles,
passes the harness unit tests, and does not reduce success on the
regression suite.

ROUTED TASKS
{for each task: statement, trajectories, p_before, p_after, delta}

CURRENT HARNESS SOURCE
{harness_source}
```

H.4 BASELINE-SPECIFIC CALLS

Three baselines require a prompted call that CoEvo-RSI does not.

SIA delegates the choice of which side to update to a separate feedback agent, which we instantiate with the same backbone.

```
Two updates are available this round: retraining the policy, or editing the
harness.  The trajectories collected since the last update follow.  Decide
which single update to apply this round.

Answer with exactly one word, ‘policy‘ or ‘harness‘, then one sentence of
reason.

TRAJECTORIES SINCE LAST UPDATE
{trajectories}
```

HASE lets the agent choose between solving and editing inside a single trajectory. We implement this as an additional tool appended to the task-agent manifest, so the choice is made by the same policy that solves the task.

```
You may also call:

  propose_harness_edit(diff, rationale)
      Propose a change to your own execution harness.  The edit is applied
      only if it passes the acceptance gate; if it passes, the rest of this
      attempt runs under the edited harness.  Proposing costs one step
      whether or not the edit is accepted.
```

The non-evolutionary arm spends its budget on parallel sampling with verifier selection followed by sequential refinement over a scratchpad that persists across rounds.

```
# selection over n parallel attempts
{n} independent attempts at the task below are shown, each with its final
state and the verifier's output where one is available.  Select the attempt
most likely to be correct.  Answer with an index, or ‘none‘ if none of them
is.

# sequential refinement
Your previous attempt and the verifier's result follow.  Revise the attempt.
You may keep any part that was correct.
```

```
Anything you write under NOTES is carried into the next round; everything
else is discarded.
```

I ADDITIONAL CO-EVOLUTION CURVES

Figure 6 shows the SWE-bench Pro and GAIA trajectories under the protocol of §3.3. Figure 7 compares HELIX-A, HASE, and CoEvo-RSI across all four benchmarks. Table 1 reports their final scores.

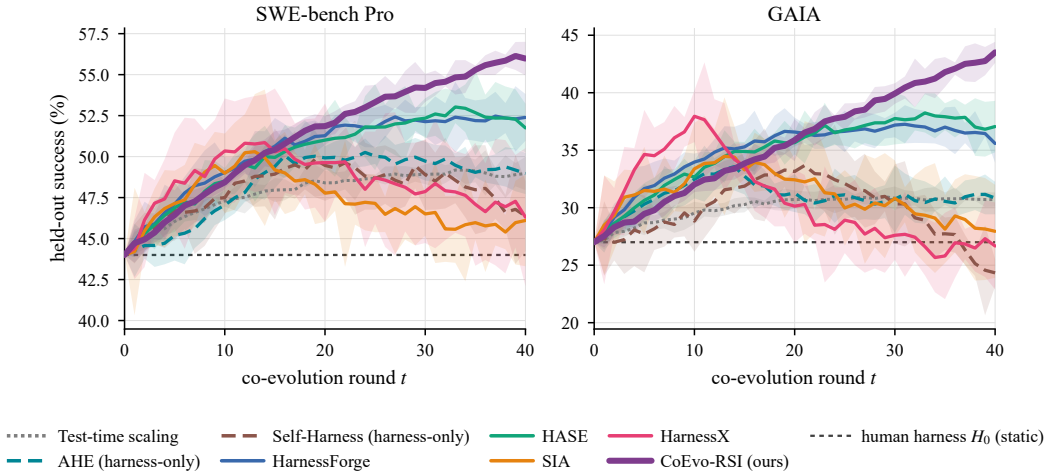


Figure 6: Held-out success over 40 co-evolution rounds on SWE-bench Pro and GAIA, five seeds, shaded 95% interval of the seed mean.

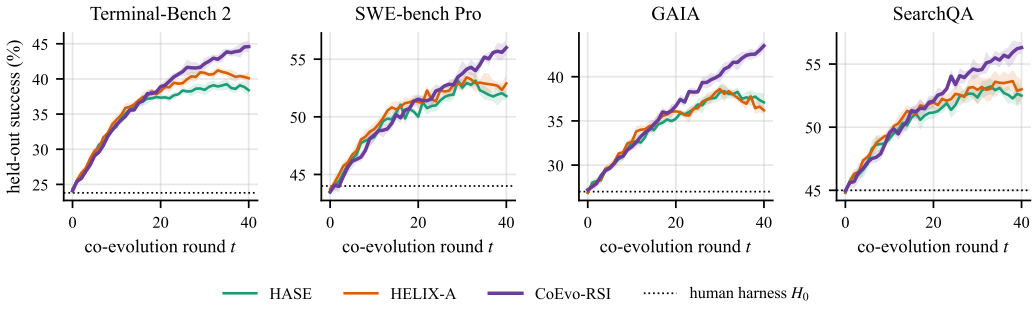


Figure 7: HELIX-A (iterated build–update–rebuild under the shared cap) against HASE and CoEvo-RSI, five seeds, shaded 95% interval of the seed mean. HELIX-A is the strongest baseline on three of four benchmarks and leads CoEvo-RSI through roughly round 18 on Terminal-Bench 2 before saturating.

J DETAILED METRICS

This appendix collects the per-round and per-metric breakdowns underlying Figures 1–4.

J.1 IMPROVEMENT-TRAJECTORY METRICS

Table 10 reports trajectory metrics across the four benchmarks. SIA, Self-Harness, and HarnessX have the largest mean drawdowns at 5.2, 5.3, and 4.3 points. CoEvo-RSI has zero drawdown on

Table 10: Improvement-trajectory metrics averaged over the four benchmarks. D_{pf} is the drawdown from peak ($\max_t y_t - y_{40}$). $\mathrm{Slope}_{20\text{–}40}$ is the least-squares slope over the second half in points per round. t_{95} is the saturation round. $\mathrm{AUC}_{1\text{–}40}$ is the mean held-out score over rounds 1–40. All columns summarise the same curve and are correlated.

Method	final	D_{pf}	$\mathrm{slope}_{20\text{–}40}$	final − R15	t_{95}	$\mathrm{AUC}_{1\text{–}40}$	$\mathrm{mean}_{36\text{–}40}$
Test-time scaling	39.8	0.3	+0.00	+0.6	23	38.6	39.7
AHE (harness-only)	40.8	1.7	−0.04	−1.1	15	39.9	40.9
Self-Harness (harness-only)	36.9	5.3	−0.24	−4.2	17	38.4	37.2
HarnessForge	44.3	0.9	+0.02	+0.9	25	41.9	44.2
HASE	44.9	0.8	+0.06	+2.2	28	42.3	44.7
SIA	38.1	5.2	−0.17	−4.4	11	38.9	38.4
HarnessX	41.2	4.3	−0.08	−2.5	14	40.6	41.5
HELIX-A (iterated)	45.6	1.0	+0.05	+2.0	27	42.6	45.4
CoEvo-RSI (ours)	**50.1**	**0.0**	**+0.26**	**+7.2**	**36**	**43.4**	**49.8**

Table 11: Allocation rules. *Source* indicates where the rule comes from. B2, B3, B4, and B7 isolate the rules assigned to our HarnessForge, HarnessX, HASE, and HELIX adaptations.

		Terminal-Bench 2				SearchQA			
Rule	Source	R10	R25	R40	D_{pf}	R10	R25	R40	D_{pf}
B1: all trajectories to both	*naive*	32.4	30.8	26.9	6.1	49.5	47.9	46.8	4.1
B0: uniform to both	*naive*	30.0	33.6	35.3	0.3	48.2	50.0	50.7	0.5
B2: success-only to model	*HarnessForge*	31.1	33.2	33.3	0.4	48.9	49.7	50.1	0.2
B3: FIFO buffer	*HarnessX*	33.7	34.7	32.1	4.1	50.2	50.9	49.0	2.9
B4: PoC gate	*HASE*	30.1	31.8	32.0	0.5	48.4	49.2	49.5	0.1
B5: failure→harness	*human intuition*	33.6	34.7	32.5	2.6	49.9	50.0	49.5	1.8
B6: entropy sampling	*active learning*	32.1	37.5	40.0	0.4	49.5	52.3	53.9	0.1
B7: verified-record curation	*HELIX*	31.8	35.9	37.4	0.3	49.0	51.4	52.1	0.2
CoEvo-RSI (ours)	*this work*	**32.9**	**40.9**	**44.6**	**0.0**	**49.1**	**53.5**	**56.3**	**0.0**

Table 12: Balancing paradigms and the ablations of CoEvo-RSI on Terminal-Bench 2. σ_Δ is the standard deviation of the round-to-round change and *worst drop* the largest single-round decline, both averaged over seeds.

Method	Balancing paradigm	final	D_{pf}	t_{95}	σ_Δ	worst drop	collapse
SIA	external arbiter	29.7	6.7	10	3.4	−6.7	yes (R15)
HarnessForge	strict alternation	37.6	0.7	23	1.6	−3.3	no
FST	fixed tempo	36.5	1.3	23	1.5	−2.8	no
HASE	implicit, within-trajectory	38.4	0.5	24	1.8	−3.6	no
Ablations of CoEvo-RSI							
w/o selection (all trajectories to both)	no selection	26.9	6.1	12	3.8	−8.2	yes (R13)
w/o Self-Judgment ($\beta \equiv 0$)	Δ only, no routing	29.0	7.1	14	4.8	−10.3	yes (R14)
w/o adaptive β ($\beta = \bar{\beta}$)	signal-driven, fixed trust	40.4	0.8	27	2.3	−4.6	no
CoEvo-RSI (full)	signal-driven, adaptive trust	**44.6**	**0.0**	**35**	**1.1**	**−2.1**	no

three benchmarks and 0.2 on SWE-bench Pro. AHE, Self-Harness, SIA, and HarnessX decline between rounds 20 and 40, while HarnessForge, HASE, and HELIX-A improve slowly at +0.02 to +0.06 points per round. CoEvo-RSI improves at +0.26 and gains 7.2 points after round 15. Its area under the curve exceeds HELIX-A by 0.8 points because its lead emerges near round 18. The last-five-round mean is 49.8, compared with 45.4 for HELIX-A. Appendix S extends the analysis to 80 rounds on Terminal-Bench 2.

J.2 Predicted-Attribution Details

The fraction of failed held-out tasks that *Self-Judgment* attributes to the harness falls from 74.0 to 42.3 on SWE-bench Pro and from 79.0 to 46.8 on GAIA, complementing the Terminal-Bench 2

result in Figure 5. GAIA has the largest post-edit rebound at 10.8 points, while SearchQA has the smallest at 3.2 points and remains above the equal-split line at round 40 with 52. These are predicted labels. On the random calibration set of Appendix N, the fraction of failures whose matched interventions favour the harness falls from 64% to 45% over the same horizon, so the counterfactual share moves in the same direction but by less than the predicted share, and part of the predicted migration is a shift in the judge's decision threshold and not a change in which intervention helps. For the weakest backbone, *Self-Judgment* intervention success stays between 52 and 62 and $\beta_t < 0.13$. Across the main run, *Self-Judgment* contributes 19, 47, and 71 percent of routing mass at rounds 5, 20, and 40, which is the handoff described by the mixture of Eq. 8.

Mean per-task Δ falls from 0.22 at round 1 to 0.04 at round 40. Around round 22, attribution overtakes change in routing mass. This diagnostic complements Figure 5 showing increasing attribution weight and shrinking measured changes jointly shift allocation toward the predicted receiver, as described by Eq. 8.

J.3 Allocation-Rule Details

Table 11 provides per-round values for both benchmarks in Figure 3 and adds verified-record curation (B7, HELIX), and the pattern is the same on both. Every naive or conservative rule from B0 to B5 either plateaus or gives back part of its gain by round 40, with drawdowns from 0.1 to 6.1; B7, the verified-record curation of HELIX, is stable and saturates between B2 and B6, only B6 and CoEvo-RSI are still improving at the final round. CoEvo-RSI does not lead at round 10 on either benchmark (32.9 on Terminal-Bench 2 against 33.7 for B3 and 33.6 for B5; 49.1 on SearchQA against 50.2 for B3). By round 25 it has overtaken every alternative and it extends the lead through round 40; the margin over B6 is 4.6 points on Terminal-Bench 2 and 2.4 on SearchQA, with task-level intervals in Table 8. Appendix L examines how much of B6's strength the uncorrected magnitude Δ^{raw} shares because it is partly an uncertainty score.

J.4 Balancing-Paradigm Details

Table 12 reports the balancing paradigms and the ablations. HarnessForge and FST finish at 37.6 and 36.5, SIA at 29.7 with $\sigma_\Delta = 3.4$, and HASE is strongest of the four at 38.4. CoEvo-RSI trails HASE by 0.4 points at round 10, leads by 0.8 at round 15, and by 2.4, 4.7, and 6.2 points at rounds 20, 30, and 40. SIA collapses at round 15 under an externally arbitrated schedule. Removing selection entirely collapses at round 13 with a worst-case drop of -8.2, and removing *Self-Judgment* ($\beta \equiv 0$) collapses at round 14 with the largest drop in the table (-10.3) despite keeping *Behavioral-Change*, so Δ alone does not prevent the two components from repeatedly reacting to the same tasks. The three paradigms that avoid collapse (HarnessForge, FST, HASE) constrain *when* an update may happen and show σ_Δ between 1.5 and 1.8. CoEvo-RSI reaches $\sigma_\Delta = 1.1$ without constraining the schedule, by constraining which tasks an unrestricted schedule acts on. Replacing adaptive β_t with its time-average neither collapses nor matches the full method (40.4 versus 44.6, $\sigma_\Delta = 2.3$ versus 1.1), and Appendix K shows how much of that gap a fixed time schedule recovers.

K β Schedules, Feedback Controls, and Negative Controls

The adaptive rule produces a low β early and a high β late, so a fixed increasing schedule might capture most of its benefit. Table 13 tests this directly. There are three findsings.

First, no constant β recovers the adaptive schedule. The best constant is $\bar{\beta} = 0.18$ at 40.4 against 44.6, with degradation in both directions and visible instability at $\beta = 0.50$.

Second, increasing time schedules recover most of the gap on the strong backbone. A linear or cosine ramp reaches 42.6–42.9, and the oracle replay of the adaptive run's own β_t trajectory reaches 43.8, which is 0.8 points below the adaptive rule and not distinguishable from it at the task level (Table 8). A shuffled-q control, which computes β_t from a random permutation of the observed feedback sequence and therefore has the same marginal distribution of β without its temporal alignment, falls to 40.9, the constant-β level. On this backbone the value of the adaptive rule is mostly that it finds a good temporal profile, and the feedback itself adds little beyond a well-chosen ramp.

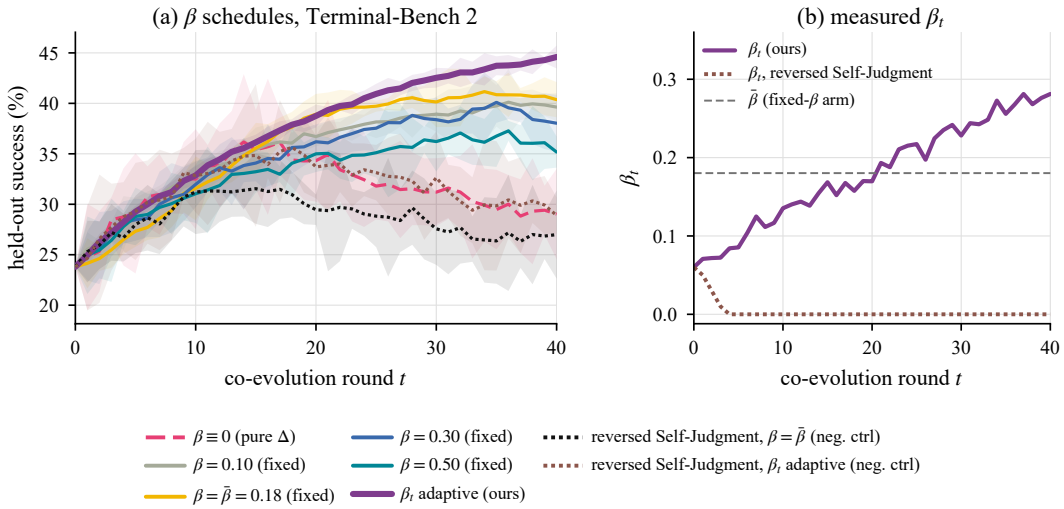


Figure 8: (a) Fixed versus adaptive β, and the two reversed-*Self-Judgment* negative controls, on Terminal-Bench 2. (b) The measured β_t schedule. Under reversed *Self-Judgment* the measured intervention success rate falls below 50%, so β_t is driven to zero within four rounds and the system reverts to pure Δ routing.

Table 13: β rules on Terminal-Bench 2 (three seeds). The time schedules use no feedback. The oracle replay fixes β_t to the seed-averaged trajectory measured in the adaptive run, so it is the best time schedule available to this backbone. The last column runs the same rule on Qwen3.5-9B, whose judge is less reliable.

		Qwen3.5-35B-A3B				Qwen3.5-9B
β rule	family	final	D_{pf}	σ_Δ	t_{95}	final
$\beta \equiv 0$ (pure Δ)	constant	29.0	7.1	4.8	14	15.9
$\beta = 0.10$	constant	39.6	0.5	2.1	33	18.4
$\beta = \bar{\beta} = 0.18$	constant	40.4	0.8	2.3	27	17.6
$\beta = 0.30$	constant	38.0	2.1	2.6	33	15.1
$\beta = 0.50$	constant	35.2	2.1	3.1	30	13.3
linear ramp $0 \to 0.30$	time schedule	42.6	0.6	1.7	33	19.5
cosine ramp $0 \to 0.30$	time schedule	42.9	0.5	1.6	33	19.2
delayed adaptive (β_{t-5})	time schedule	43.1	0.4	1.5	34	21.7
oracle replay of the adaptive $\bar{\beta}_t$ trajectory	time schedule	43.8	0.3	1.3	34	19.1
shuffled-q (permuted feedback sequence)	feedback control	40.9	1.9	2.5	29	17.0
$\beta_t = \beta_{\max} \max(0, 2q_{t-1} - 1)$ **(ours)**	feedback	**44.6**	**0.0**	**1.1**	**35**	**23.4**
β_t from calibrated Youden index J^{cal}_{t-1}	feedback	44.9	0.0	1.2	34	23.7
reversed Self-Judgment, $\beta = \bar{\beta}$ (neg. ctrl)	control	27.0	4.6	4.1	10	12.8
reversed Self-Judgment, β_t adaptive (neg. ctrl)	control	28.9	6.7	4.7	17	15.6

Third, the feedback matters when the judge's reliability differs from the one the schedule was tuned on. On Qwen3.5-9B, whose calibrated attribution accuracy stays near 57%, the ramps and the replayed strong-backbone schedule over-trust the judge and finish 2–4 points below the adaptive rule (19.1–19.5 versus 23.4), while the delayed adaptive rule, which uses feedback with a lag, loses 1.7. The prevalence-aware coefficient driven by the calibrated Youden index (Appendix N) performs the same as the executed-intervention rule on both backbones (44.9 and 23.7), so the cheaper statistic is an adequate proxy for the calibrated one in this setting.

If *Self-Judgment* is inverted with the coefficient held fixed at $\bar{\beta}$, the final score falls to 27.0, below the $\beta \equiv 0$ arm. If it is inverted while β remains adaptive, the measured intervention success drops below 50% within the first rounds, β_t clamps to zero by round 4, and the run tracks the $\beta \equiv 0$ arm (28.9 versus 29.0). These controls verify that the rule turns off when its feedback says it should

and that the attribution has a directional effect. The matched-intervention calibration of Appendix N tests whether the attribution is causally correct.

L Measurement Budget, Null Correction, and Uncertainty Sampling

With N rollouts per state, the success estimate moves in steps of $1/N$. Table 3 shows that the success component of the raw magnitude is nonzero 72.7% of the time at $N = 4$ and 80.4% at $N = 8$ under no true change. Its mean magnitude $e_0(1/2)$ falls from 0.273 at $N = 4$ to 0.196 at $N = 8$ and 0.140 at $N = 16$. This sampling noise is largest near $p = 1/2$, where entropy sampling also concentrates. Table 14 evaluates routing quality as N changes and compares the null-corrected score of Eq. 5 with the uncorrected magnitude Δ^{raw} on Terminal-Bench 2 across three seeds.

Table 14: Measurement-budget sweep on Terminal-Bench 2 (three seeds). Budget-matched rows charge measurement to the round cap, which reduces the realized training iterations. The uncharged rows give measurement for free. Δ is the null-corrected score of Eq. 5 and Δ^{raw} the uncorrected magnitude. The residualized score replaces $|\hat{D}|$ in Δ^{raw} by its positive residual after regressing on $\bar{p}(1-\bar{p})$ across the panel. The entropy rows use the default $N = 8$.

Measurement rollouts per state N	4	8 (default)	16	32
measurement share of the round cap (Terminal-Bench 2)	2.4%	4.8%	9.7%	19.3%
realized GRPO iterations, budget-matched	15	14	13	12
final, null-corrected Δ, budget-matched	42.6	**44.6**	44.9	44.1
final, null-corrected Δ, measurement not charged	42.6	45.0	46.0	46.3
final, raw magnitude Δ^{raw}, budget-matched	43.9	44.5	44.4	43.6
final, raw magnitude, measurement not charged	43.9	44.9	45.5	45.9
final, Δ^{raw} residualized on entropy, budget-matched	43.1	44.3	44.7	44.0
Top-K Jaccard overlap, two independent measurement draws	0.52	0.78	0.88	0.88
Spearman ρ of Δ with a 64-rollout reference	0.85	0.95	0.97	0.98
Pearson r of Δ^{raw} with $\bar{p}(1-\bar{p})$, rounds 1–5 / 36–40	0.31 / 0.52	0.26 / 0.44	0.19 / 0.33	0.14 / 0.24
Pearson r of Δ with $\bar{p}(1-\bar{p})$, rounds 1–5 / 36–40	0.12 / 0.19	0.10 / 0.15	0.08 / 0.11	0.06 / 0.09
B6 entropy sampling		40.0		
B6 entropy $+\beta_t$ Self-Judgment		42.0		

Under a fixed cap, the final score peaks between $N = 8$ and $N = 16$ (44.6 and 44.9) and declines at $N = 32$ (44.1), where measurement takes 19% of the cap and training loses three iterations per round. With measurement uncharged, the score rises monotonically to 46.3 at $N = 32$, so the decline reflects the training budget that measurement consumes. Routing stability improves with N and levels off near $N = 16$. The Jaccard overlap of Top-K harness selections between two independent measurement draws rises from 0.52 to 0.88, which corresponds to 11 to 15 of the 16 selected tasks being shared, and the rank correlation of the panel scores with a 64-rollout reference rises from 0.85 to 0.98. The default $N = 8$ uses 4.8% of the cap for measurement, half of the share at $N = 16$, and lies within 0.3 points of the best matched-cap score while sharing 14 of the 16 selected tasks between two draws against 15 at $N = 16$ and $N = 32$. At $N = 4$ the correction costs 1.3 points relative to the raw magnitude because it clips real changes of one rollout in four together with the noise. From $N = 8$ on, the corrected score matches or exceeds the raw one.

The correlation of Δ^{raw} with the entropy score $\bar{p}(1-\bar{p})$ at $N = 8$ rises from 0.26 in rounds 1–5 to 0.44 in rounds 36–40, because true changes shrink late in the run while the null magnitude does not. The corrected score correlates at 0.10 and 0.15 in the same blocks. Entropy sampling (B6) alone reaches 40.0, and adding the same β_t *Self-Judgment* term raises it to 42.0, so attribution accounts for 2.0 of the 4.6 points of margin over B6. The corrected score reaches 44.6 at $N = 8$, while an entropy-residualized raw score reaches 44.3. The measured change signal therefore accounts for the remaining 2.6 points. Null correction reduces the uncertainty component, and the corrected $N = 8$ setting balances measurement and training under the matched cap.

M SIGNED CHANGE AND SAME-SIDE REPAIR

The magnitude in Eq. 5 treats an improvement and a regression alike and hands both to the counterpart. This is a design choice with a specific alternative. A regression caused by a model update could be repaired by further model training or by reverting the update, and a regression caused by a harness edit could be reverted. The harness side already has a same-side revert, because the acceptance gate rejects edits that regress the regression suite, and the harness evolver can also delete earlier mechanisms (Case 6). The model side has none by default. Table 15 compares the alternatives on Terminal-Bench 2 with three seeds.

Table 15: Regression handling on Terminal-Bench 2 (three seeds). R3 routes a task whose success fell after a model update back into the model's training distribution, and a task whose success fell after a harness edit to the evolver with a revert option. Neither goes to the counterpart. R4 reverts an update whose panel-mean intervention gain $\hat{G}$ is below $-\tau$ with $\tau = 0.01$, one point of success, and then applies R0.

Rule	what a regression does	final	D_{pf}	σ_{Δ}
R0: magnitude Δ, counterpart receives (ours)	counterpart inspects it	**44.6**	0.0	1.1
R1: improvements only	ignored	42.2	0.5	1.4
R2: regressions only	counterpart inspects it	39.8	1.8	2.2
R3: regressions routed to the side that caused them (same-side repair)	causing side repairs it	43.9	0.3	1.2
R4: R0 + revert-on-aggregate-regression ($\tau = 1.0$ point)	update reverted, then R0	45.0	0.0	1.0
R5: signed score (improvements to counterpart, regressions to cause)	split by sign	43.1	0.6	1.3
R4 reverts triggered per run (mean over seeds), TB2 / GAIA		3.0 / 2.4 of 80 updates		

Improvements-only (R1) and regressions-only (R2) each lose part of the signal, with regressions alone losing more, because most late-round changes are improvements. Same-side repair (R3) is competitive (43.9) and does not beat counterpart routing. Inspecting the routed tasks shows why. When a model update regresses a task, the harness-side evolver frequently identifies a harness mechanism that the new policy's behaviour no longer fits, as in Case 6, and repairing the policy instead re-learns the old behaviour. Aggregate revert-on-regression (R4) is cheap, triggers about three times in 80 updates, and adds 0.4 points over R0, it is compatible with the method and a sensible safety rule, and the difference is within seed variation. Splitting by sign (R5) underperforms both R0 and R3, consistent with the routed-task inspection. Which side should act on a regression is not determined by which side caused it.

N ATTRIBUTION CALIBRATION WITH MATCHED INTERVENTIONS AND JUDGE CONTROLS

The executed-intervention statistic q_t counts a *Self-Judgment* prediction as successful when the intervention on the predicted side improves the task. It has no counterfactual, since both interventions might have helped or the other one might have helped more, and it is measured only on tasks the router chose. This appendix adds the missing counterfactual and the missing randomisation.

Each round, 16 failed tasks are drawn uniformly from the tasks of the current panel that fail at the state (θ_{t+1}, H_t) measured in step (iii), independently of the router. Their starting-state success and *Self-Judgment* label are those of that measurement and are not obtained again. From this state each task receives two interventions on separate branches that are discarded afterwards. The model-only branch runs two GRPO iterations on that task (2×8 rollouts) and is evaluated with $N = 4$ under H_t. The harness-only branch makes one evolver call using only that task's trajectories, applies the edit if it passes the path guard, the safety linter, compilation, and the unit tests, and is evaluated with $N = 4$ under θ_{t+1}. The regression suite is not applied to discarded branches because they are never committed, and a branch that fails a gate leaves the harness unchanged, so its gain is zero. The counterfactual label is $\ell(x) =$ HARNESS when $G^{\mathrm{H}} > G^{\mathrm{M}} + 0.05$, MODEL when $G^{\mathrm{M}} > G^{\mathrm{H}} + 0.05$, and a tie otherwise. The step costs $16 \times (16 + 4 + 4) = 384$ rollouts and 16 evolver calls per round, charged to the cap (Table 4), and it uses all failed panel tasks when fewer than 16 are available. Small interventions and $N = 4$ measurements make the labels noisy, so calibrated accuracy is reported against this operational reference.

Table 16: Self-Judgment against counterfactual labels on the random calibration set, Terminal-Bench 2, Qwen3.5-35B-A3B, with counts averaged over the five seeds and rounded to integers. Each seed contributes 160 calibration tasks per block, 16 per round over 10 rounds, and 800 across seeds, and ties are excluded from the confusion matrix. Accuracy and the Youden index are computed from the averaged matrix. The last row compares executed-intervention success on the routed set with the same statistic on the random set.

	rounds 1–10		rounds 16–25		rounds 31–40	
counterfactual label $\ell(x) \rightarrow$	harness	model	harness	model	harness	model
Self-Judgment predicts harness	52	23	45	22	45	20
Self-Judgment predicts model	31	23	24	40	16	56
ties excluded ($\lvert G^{\mathrm{M}} - G^{\mathrm{H}} \rvert < 0.05$)	31 of 160		29 of 160		23 of 160	
accuracy / Youden J^{cal}	58.1% / 0.13		64.9% / 0.30		73.7% / 0.47	
counterfactual harness share	64%		53%		45%	
predicted harness share (same tasks)	58%		51%		47%	
executed-intervention success q_t, routed set / random calibration set	56% / 55%		68% / 63%		78% / 71%	

Calibrated accuracy rises from 58% in rounds 1–10 to 74% in rounds 31–40, and the prevalence-independent Youden index from 0.13 to 0.47, so the improvement in q_t reported in §3.6 is more than a selection effect and the judge's comparative attribution becomes more reliable. The selection effect is measurable, and executed-intervention success is 78% on the routed set and 71% on the random set in the last block, a 7-point inflation that the calibrated coefficient does not inherit. The true-positive and true-negative rates for the harness label are 65.2% and 64.5% in rounds 16–25 and 73.8% and 73.7% in rounds 31–40, so $2\,\mathrm{acc} - 1$ matches the Youden index in those blocks. In rounds 1–10 they are 62.7% and 50.0%, and $2\,\mathrm{acc} - 1 = 0.16$ against $J^{\mathrm{cal}} = 0.13$. Running β_t from J^{cal} in place of q_t changes the final score by 0.3 (Table 13). Ties are 14–19% of calibration tasks. On these, both interventions help by similar amounts or neither does, and a binary judge cannot be right or wrong about them.

Because β_t raises the weight of tasks the judge already labels, q_t could rise because the router increasingly selects tasks on which its predicted intervention happens to work. The random calibration set breaks this loop. Its accuracy rises by 16 points over the run, and the 7-point routed-versus-random gap is the size of the loop's contribution to q_t.

The main configuration uses the task model as its own judge, so rising accuracy could reflect the model becoming stronger. Table 17 substitutes fixed judges. A frozen copy of the round-0 task model starts at the same 58% and rises to 66%, an improvement of 8 points against 16 points for the evolving self-judge. Each judge condition evolves its own run, so routing and the distribution of failures differ between conditions, and the comparison does not apportion the difference between the model's growth and the failures becoming easier to attribute as the harness matures. A frozen Qwen3.5-122B-A10B judge is more accurate throughout and improves the final score by 0.7. A frozen Qwen3.5-9B judge is close to chance, the adaptive coefficient stays below 0.07, and the final score falls to 31.8, which is 2.8 points above the $\beta \equiv 0$ arm because a near-zero β leaves Δ routing largely intact. Across the four judges tested, the final score is monotone in the judge's calibrated accuracy.

Table 17: Judges and label sets on Terminal-Bench 2 (three seeds). ρ^{cal} is calibrated accuracy on the random calibration set. Human labels were collected for 120 failures at rounds 5, 20, and 40 from three annotators with access to the trajectory and the harness source. The annotators were three of the authors.

Judge	type	final	$\rho^{\mathrm{cal}}_{1\text{–}10} \rightarrow \rho^{\mathrm{cal}}_{31\text{–}40}$	β_{40}	human agr. (R5→R40)
task model (self, default)	evolving	**44.6**	58% → 74%	0.28	63% → 72%
frozen round-0 copy of the task model	fixed, round-0 copy	44.0	58% → 66%	0.21	63% → 66%
frozen Qwen3.5-122B-A10B	fixed, stronger	45.3	66% → 79%	0.34	70% → 76%
frozen Qwen3.5-9B	fixed, weaker	31.8	52% → 57%	0.06	54% → 58%
Four-way attribution with the task model as judge					
{harness, model, both, unsure}	evolving	44.9	60% → 75%	0.29	—
label shares at round 40: harness / model / both / unsure		34% / 47% / 11% / 8%			
human–human Fleiss κ (120 failures, 3 annotators)		0.66			
human majority vs. counterfactual label $\ell(x)$		71%			

Three of the authors labelled 120 failures, 40 drawn uniformly at each of rounds 5, 20, and 40. They received the judge prompt of Appendix H with the harness source added and labelled agent trajectories only, so no personal data was collected and no human-subjects procedure applied. Their agreement is moderate (Fleiss $\kappa = 0.66$ (Fleiss, 1971)), their majority agrees with the counterfactual label 71% of the time, and the self-judge's agreement with the human majority rises from 63% to 72% over the run. Attribution is an uncertain judgment even for people with more information than the judge receives.

Replacing the binary answer set with {HARNESS, MODEL, BOTH, UNSURE}, routing BOTH to both receivers and giving UNSURE no attribution weight, changes the final score by $+0.3$ and moves 19% of labels into the two new classes at round 40. Breaking majority-vote ties toward HARNESS in place of MODEL changes the final score by -0.4 and the round-40 predicted harness share by $+3$ points. Neither change is distinguishable from seed variation, so the binary interface does not limit the method at this scale, and the four-way variant is the better-specified interface.

O Model–Harness Compatibility Matrix

The introduction argues that a harness evolved against one model becomes mismatched with another, and vice versa. §3.2 examines a related situation, in which the checkpoints come from one co-evolved run and both components change. Here we give the full matrix and its reading.

Table 18: Held-out success on Terminal-Bench 2 for every pairing of model and harness checkpoints from the same run (three seeds, every cell is evaluated with the same 356 held-out rollouts). Diagonal cells are the states the run actually visited. The deficit of an off-diagonal cell is measured against the better of its two matched cells.

	CoEvo-RSI checkpoints				HASE checkpoints			
	H_0	H_{13}	H_{26}	H_{40}	H_0	H_{13}	H_{26}	H_{40}
θ_0	**23.8**	26.1	27.4	27.9	**23.8**	25.9	26.6	27.1
θ_{13}	29.5	**35.4**	34.6	33.0	28.7	**33.9**	33.8	33.4
θ_{26}	32.0	37.7	**40.8**	39.6	30.4	35.6	**37.0**	36.8
θ_{40}	34.2	38.3	42.9	**44.6**	31.6	36.1	38.0	**38.4**
mean deficit vs. the better matched pair, distance 1 / 2 / 3	5.2 / 10.0 / 13.6				3.3 / 6.1 / 9.0			

Three patterns are visible. First, off-diagonal cells are below the better matched cell in every case, and the deficit grows with the distance between checkpoints, from 5.2 points one step apart to 13.6 three steps apart for CoEvo-RSI. The better matched cell is also the later checkpoint, so this deficit includes ordinary training progress. Second, the asymmetry is the one the introduction predicts. Rows for intermediate models peak at their matched harness and decline with later harnesses (θ_{13}: $35.4 \rightarrow 34.6 \rightarrow 33.0$; θ_{26}: $40.8 \rightarrow 39.6$), so a more evolved harness is worse for an earlier model, because later edits assume behaviours the earlier model has not learned, such as the restrained use of the log-search tool in Case 1 or the verbatim search strings of Case 6. Columns increase monotonically with the model checkpoint, so a later model is never hurt by an earlier harness, only under-served. Third, the same matrix for HASE has a smaller deficit (3.3/6.1/9.0) because its harness changes less over the run. Less co-adaptation is one reason its endpoint is lower. The in-domain 2×2 interaction of Appendix R ($+6.3$ for CoEvo-RSI, $+3.5$ for HASE) is the corner of this matrix.

Pairing the model from one seed with the same-round harness from another retains 84% of the matched gain at every stage. The gap from the same-seed pairing grows from 1.8 to 3.4 points over the run. This comparison separates compatibility associated with the training stage from compatibility specific to an optimisation path. Most of the gain survives a same-stage swap, and a smaller part depends on the particular model–harness history.

P Different Model Scales and Initial Harnesses

The main experiments fix one backbone and one initial harness. This appendix varies each in turn with three seeds, holding the rest of the protocol identical.

Across a range of more than ten-fold in total parameters CoEvo-RSI improves over the strongest co-evolution baseline on every backbone, and the margin shrinks with capability, from $+7.4$ points on

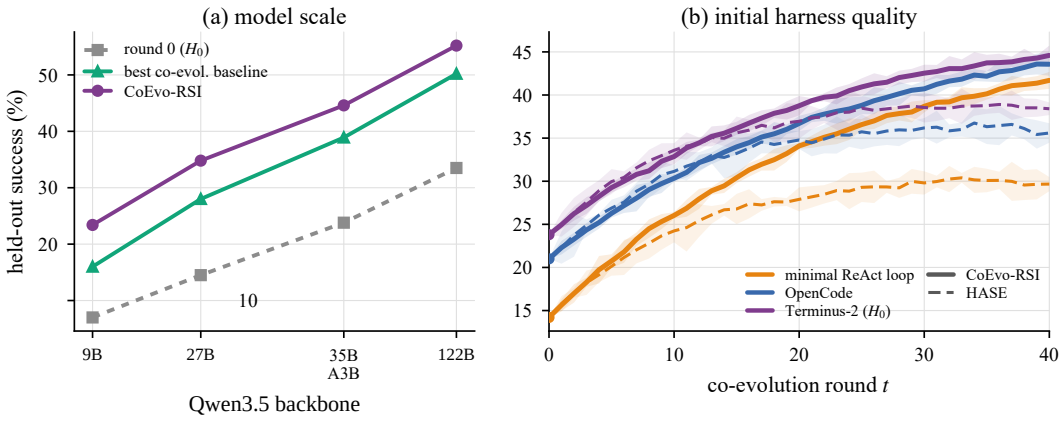


Figure 9: (a) Round-40 held-out success on Terminal-Bench 2 across the Qwen3.5 scale sweep. (b) Co-evolution started from three initial harnesses of differing quality with the same backbone and protocol. Solid curves are CoEvo-RSI and dashed curves HASE, coloured by H_0.

Table 19: Terminal-Bench 2 at round 40 across backbones, mean ± standard deviation over three seeds. The first four rows vary scale within one family, and the last two are other families at comparable capability. $\Delta_{\text{co-ev}}$ is the margin over the strongest co-evolution baseline for that backbone, Δ_β the cost of replacing β_t with the constant $\bar{\beta}$, and ρ_{40}^{cal} the calibrated attribution accuracy in the last block.

Backbone	R0	Self-H.	best co-ev.	w/o ad. β	CoEvo-RSI	$\Delta_{\text{co-ev}}$	Δ_β	ρ_{40}^{cal}	β_{40}
Qwen3.5-9B	7.0	9.4 ±0.8	16.0 ±1.3 (HarnessX)	17.6 ±1.5	**23.4** ±1.1	+7.4	+5.8	57%	0.07
Qwen3.5-27B	14.5	18.2 ±1.0	28.0 ±1.4 (HELIX-A)	29.9 ±1.2	**34.8** ±1.0	+6.8	+4.9	64%	0.16
Qwen3.5-35B-A3B	23.8	28.9 ±1.2	40.1 ±1.1 (HELIX-A)	40.4 ±1.3	**44.6** ±0.9	+4.5	+4.2	74%	0.28
Qwen3.5-122B-A10B	33.5	39.1 ±1.1	50.6 ±1.2 (HELIX-A)	51.8 ±1.0	**55.2** ±0.8	+4.6	+3.4	79%	0.33
GLM-5	52.4	55.6 ±0.9	64.5 ±1.0 (HELIX-A)	66.0 ±0.9	**68.4** ±0.7	+3.9	+2.4	81%	0.34
MiniMax M2.5	42.2	48.6 ±1.0	62.7 ±1.1 (HELIX-A)	64.0 ±1.0	**66.3** ±0.8	+3.6	+2.3	83%	0.36

Qwen3.5-9B to +3.6 on MiniMax M2.5. The absolute final score rises with scale while the relative advantage falls, which is the pattern reported for harness evolution generally. A weaker policy has a larger share of its failures attributable to execution, so there is more for allocation to recover. The seed standard deviations are about one point, so the margins at every scale exceed two standard deviations, and the differences between adjacent margins (for example +4.5 against +4.6) do not.

Calibrated attribution accuracy rises from 57% on Qwen3.5-9B to 83% on MiniMax M2.5, and β_{40} rises with it from 0.07 to 0.36. The cost of removing adaptivity is largest on the weakest backbone (+5.8 points on Qwen3.5-9B against +2.3 on MiniMax M2.5). A weak judge's accuracy is close to the feedback threshold, so a constant β injects misrouting that the adaptive rule detects and suppresses. Adaptive β is most valuable where *Self-Judgment* is least trustworthy, which is also what the schedule study of Appendix K shows. Because β stays small on weak backbones while the gains there are large, most of the benefit on Qwen3.5-9B and Qwen3.5-27B comes from Δ and less from *Self-Judgment*.

Figure 9(b) and Table 21 repeat the Terminal-Bench 2 experiment from three starting points spanning 9.6 points, namely a minimal ReAct loop with a single shell tool and no memory management, the OpenCode harness, and Terminus-2. The spread across starting points falls from 9.6 points at round 0 to 2.9 at round 40 under CoEvo-RSI, and only to 8.7 under HASE. The run started from the minimal loop gains 27.5 points and ends within 2.9 of the run started from Terminus-2, overtaking the Terminus-2 HASE run by round 30. A conservative filter inherits the quality of what it is given. HASE accepts only edits with an immediate measured gain, and from a weak harness most individually promising edits have no immediate effect until several are in place, so the run stalls near its starting point. Re-measuring Δ after every update makes the size of a deficit visible in proportion to how much behaviour it changes, so a large early deficit produces a correspondingly

Table 20: Checkpoints and training configuration of every backbone. Total and active parameters follow the model pages. Trainable parameters are rank-64 LoRA adapters on the attention and dense or shared-expert MLP projections, with the base weights, routed experts, and vision encoder frozen.

Backbone	Hugging Face identifier	checkpoint	parameters, total / active	trainable	H100 GPUs
Qwen3.5-9B	`Qwen/Qwen3.5-9B`	post-trained	10B / 10B	0.20B	8
Qwen3.5-27B	`Qwen/Qwen3.5-27B`	post-trained	28B / 28B	0.32B	8
Qwen3.5-35B-A3B	`Qwen/Qwen3.5-35B-A3B`	post-trained	36B / 3B	0.30B	8
Qwen3.5-122B-A10B	`Qwen/Qwen3.5-122B-A10B`	post-trained	125B / 10B	0.56B	16
MiniMax M2.5	`MiniMaxAI/MiniMax-M2.5`	post-trained	229B / —	0.70B	16
GLM-5	`zai-org/GLM-5`	post-trained	744B / 40B	1.10B	64

Table 21: Initial-harness-quality sweep on Terminal-Bench 2 with Qwen3.5-35B-A3B (three seeds). The final row is the spread across the three initial harnesses at that column.

Initial harness H_0	R0	HASE R40	CoEvo-RSI R40	CoEvo-RSI gain
minimal ReAct loop	14.2	29.7 $\pm$1.4	**41.7** $\pm$1.2	+27.5
OpenCode	21.0	35.6 $\pm$1.2	**43.6** $\pm$1.0	+22.6
Terminus-2 (H_0)	23.8	38.4 $\pm$1.1	**44.6** $\pm$0.9	+20.8
spread (max − min)	9.6	8.7	2.9	

large routing weight. The practical implication is that the engineering effort invested in H_0 is largely recovered over a long run.

Q The Allocation Rule inside Other Systems

Appendix P varies the harness the method starts from. This section varies the system the rule is embedded in. CoEvo-RSI is a rule for constructing $\mathcal{T}^{\text{model}}$ and $\mathcal{T}^{\text{harness}}$, and that rule is separable from the rest of the loop, so it can replace the allocation step of another self-improving system. We take seven systems, keep their update schedule, acceptance gate, optimiser, and population or buffer machinery exactly as adapted for Table 1, and replace only the step that decides which trajectories each side receives (three seeds).

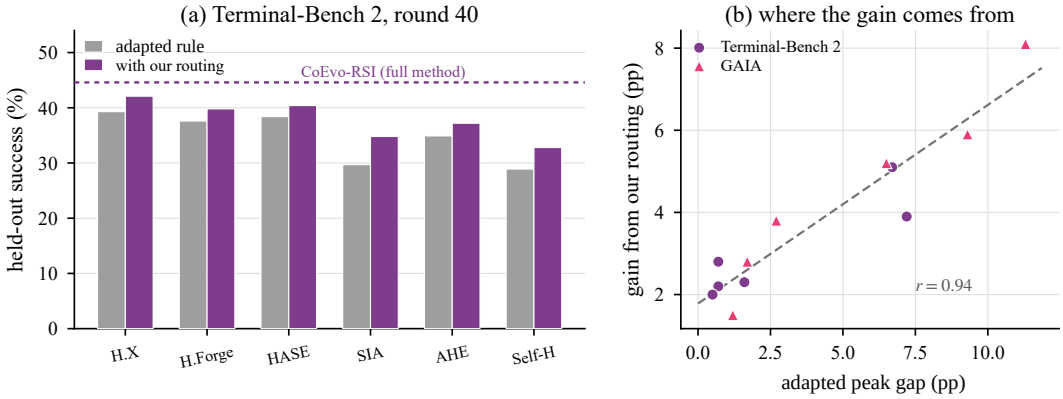


Figure 10: (a) Six host systems with their adapted allocation rule and with ours substituted on Terminal-Bench 2 at round 40. The seventh host, HELIX-A, appears in Table 22. The dashed line is CoEvo-RSI. (b) Substitution gain against the host's adapted drawdown, pooled over both benchmarks.

Substitution improves every host on both benchmarks, by +2.9 points on average on Terminal-Bench 2 and +4.3 on GAIA, and every host with our rule remains below the full method. The best, HarnessX with our routing, ends at 42.1 against 44.6. Figure 10(b) plots the gain from substitution against the host's adapted drawdown. The two are strongly related, with $r = 0.94$ across the twelve host–benchmark pairs plotted and $r = 0.95$ across all fourteen pairs of Table 22. Hosts that were

Table 22: Substituting our allocation rule into seven host systems. †Harness-only systems, where the weights are frozen.

		Terminal-Bench 2			GAIA		
Host system	Adapted allocation rule	adapted	+ours	D_{pf}	adapted	+ours	D_{pf}
HarnessX	*FIFO buffer*	39.3	**42.1**	$0.7 \to 0.5$	26.7	**34.8**	$11.3 \to 2.4$
HarnessForge	*success-only curation*	37.6	**39.8**	$0.7 \to 0.5$	35.6	**38.4**	$1.7 \to 0.9$
HASE	*PoC acceptance gate*	38.4	**40.4**	$0.5 \to 0.4$	37.1	**38.6**	$1.2 \to 0.7$
HELIX-A	*verified-record curation*	40.1	**42.0**	$0.9 \to 0.6$	36.2	**38.9**	$2.0 \to 1.1$
SIA	*arbiter-selected*	29.7	**34.8**	$6.7 \to 2.1$	27.9	**33.1**	$6.5 \to 1.8$
AHE†	*observability heuristic*	34.9	**37.2**	$1.6 \to 0.9$	30.8	**34.6**	$2.7 \to 1.4$
Self-Harness†	*dual-gate acceptance*	28.9	**32.8**	$7.2 \to 2.3$	24.3	**30.2**	$9.3 \to 2.6$
mean gain			+2.9			+4.3	
CoEvo-RSI (full method)	*this work*	—	**44.6**		—	**43.5**	

already stable gain little. HASE, whose proof-of-concept gate keeps it within 0.5 points of its peak on Terminal-Bench 2, gains 2.0, and HELIX-A gains 1.9. Hosts that degraded gain most, and HarnessX on GAIA moves from 26.7 to 34.8 with its drawdown from 11.3 to 2.4. The rule's contribution to an existing system is therefore mostly the removal of a degradation mode, consistent with the ablations of §3.5. AHE and Self-Harness freeze the weights, so $\Delta^{\mathrm{M}}(x) = 0$ for every task and the rule reduces to $w_{\mathrm{harness}}(x) = \beta_t \mathbb{1}[b(x) = \mathrm{harness}]$, that is, to attribution routing alone. It still improves both hosts ($+2.3$ and $+3.9$ on Terminal-Bench 2).

R Factorial Transfer of Model and Harness

A co-evolved system could improve by overfitting its harness to one benchmark, and a transfer of the whole system cannot say whether a positive transfer comes from the model, the harness, or their interaction. Table 23 evaluates the four combinations of initial and round-40 model with initial and round-40 harness, both in-domain and on a benchmark the system never saw, with no further adaptation. The interaction is $y(\theta_{40}, H_{40}) - y(\theta_{40}, H_0) - y(\theta_0, H_{40}) + y(\theta_0, H_0)$.

Table 23: Factorial transfer without adaptation (three seeds). In the transfer rows, θ_{40} and H_{40} are the source run's final model and harness, and H_0 is the target benchmark's human harness. Reference rows give full-system transfer for the compared systems.

Source → Target		(θ_0, H_0)	(θ_{40}, H_0)	(θ_0, H_{40})	(θ_{40}, H_{40})	interaction
In-domain (source = target)						
Terminal-Bench 2	CoEvo-RSI	23.8	34.2	27.9	**44.6**	+6.3
Terminal-Bench 2	HASE	23.8	31.6	27.1	**38.4**	+3.5
GAIA	CoEvo-RSI	27.0	34.9	31.2	**43.5**	+4.4
Within-domain transfer						
Terminal-Bench 2 → SWE-bench Pro	CoEvo-RSI	44.0	46.2	45.1	**48.0**	+0.7
SWE-bench Pro → Terminal-Bench 2	CoEvo-RSI	23.8	27.6	26.9	**31.9**	+1.2
GAIA → SearchQA	CoEvo-RSI	45.0	47.4	46.6	**49.6**	+0.6
SearchQA → GAIA	CoEvo-RSI	27.0	29.8	28.7	**32.1**	+0.6
Cross-domain transfer						
Terminal-Bench 2 → GAIA	CoEvo-RSI	27.0	27.9	27.6	**28.9**	+0.4
GAIA → Terminal-Bench 2	CoEvo-RSI	23.8	25.4	24.3	**26.4**	+0.5
Reference: full-system transfer of the compared systems, target held-out set						
Terminal-Bench 2 → SWE-bench Pro	HarnessX / HASE / HELIX-A / ours	45.0 / 46.1 / 46.8 / **48.0**				
SWE-bench Pro → Terminal-Bench 2	HarnessX / HASE / HELIX-A / ours	21.9 / 28.4 / 29.0 / **31.9**				
GAIA → SearchQA	HarnessX / HASE / HELIX-A / ours	44.1 / 47.2 / 47.9 / **49.6**				
SearchQA → GAIA	HarnessX / HASE / HELIX-A / ours	27.8 / 29.6 / 29.9 / **32.1**				

In-domain, the interaction is $+6.3$ on Terminal-Bench 2 and $+4.4$ on GAIA, about 30% and 27% of the respective gains. The round-40 model with the initial harness recovers half of the total gain on Terminal-Bench 2 and the initial model with the round-40 harness one fifth, so the components contribute mostly additively and the remaining share is co-adaptation between the two, which the compatibility matrix of Appendix O shows from the inside. In transfer the interaction shrinks to $+0.4$ to $+1.2$ and the two components contribute additively. Within a domain the model carries slightly more than the harness ($+2.2$ against $+1.1$ from Terminal-Bench 2 to SWE-bench Pro),

and across domains the model contributes $+0.9$ to $+1.6$ and the harness $+0.5$ to $+0.6$. CoEvo-RSI retains roughly a third of its in-domain gain within a domain and much less across domains. HELIX-A and HASE transfer positively and by less, and HarnessX transfers negatively from the benchmarks on which it degraded. The interaction accounts for about 30% of the in-domain gain and almost none of the transferred gain, so the co-adapted part of the value is benchmark-specific and the additive parts transfer partially, mostly through the model.

S Beyond Forty Rounds

The main experiment shows CoEvo-RSI improving through round 40. Table 24 extends four arms to 80 rounds on Terminal-Bench 2 with two seeds under the same per-round cap.

Table 24: Eighty-round extension on Terminal-Bench 2 (two seeds).

Terminal-Bench 2, 80 rounds (2 seeds)	R20	R40	R60	R80	peak (round)	D_{pf} at R80	$\mathrm{slope}_{60\text{–}80}$
CoEvo-RSI	38.9	44.6	46.9	47.3	47.7 (74)	0.4	$+0.02$
HELIX-A	38.1	40.1	40.6	40.2	41.3 (58)	1.1	-0.02
HASE	36.9	38.4	38.9	38.7	39.2 (63)	0.5	-0.01
B6 entropy	36.0	40.0	41.5	41.2	41.8 (66)	0.6	-0.02

CoEvo-RSI improves through about round 60, gains 2.7 points between rounds 40 and 80, peaks at round 74, and ends 0.4 below its peak with a near-zero second-half slope. The three baselines gain at most 1.2 points after round 40, the largest gain being B6 from 40.0 to 41.2, and end 0.5 to 1.1 below their peaks. Mean per-task Δ falls to 0.03 by round 70, when attribution carries 88% of routing mass and calibrated accuracy reaches 76%. Most remaining calibration failures are ties between the two small interventions.

T Harness Complexity, Gate Outcomes, and Safety Audit

The evolver may edit prompts, tools, registrations, memory, control flow, and delegation. Compilation, unit tests, and an in-distribution regression suite establish functional behaviour and do not show that the evolved harness remains maintainable, interpretable, or safe. Table 25 reports what we measured.

Table 25: Harness complexity over rounds and gate outcomes for the Terminal-Bench 2 CoEvo-RSI runs.

Terminal-Bench 2 harness, CoEvo-RSI (mean over 5 seeds)	H_0	H_{13}	H_{26}	H_{40}
lines of code (excluding tests)	2,140	2,690	3,420	3,210
registered tools	6	9	12	11
mean cyclomatic complexity per function	3.1	3.3	3.6	3.5
functions longer than 60 lines	4	6	9	7
cumulative committed patches	0	12	27	39
cumulative net-deletion patches	0	0	3	6
Gate outcomes over 40 rounds (all seeds pooled, 233 proposed patches)				
rejected by path guard (verifier, regression suite, evaluation loop)	14			
rejected by static safety linter (network, shell escalation, workspace writes, secret reads)	9			
rejected by leakage scan (task ids, answer literals, benchmark paths)	0 (pilot without guidance: 7 of 31)			
rejected by compile, unit, or regression tests	15			
committed	195			
human audit of 40 random committed diffs: maintainable / would refactor / unsafe	34 / 6 / 0			

The harness grows from $2{,}140$ to a peak of $3{,}420$ lines at round 26 and ends 6% below the peak after net-deletion patches such as Case 6. The number of registered tools rises from 6 to 11 and mean cyclomatic complexity per function from 3.1 to 3.5. The harness therefore becomes larger and moderately more complex, and whether the same endpoint could be reached with fewer, better-chosen edits is untested.

Of 233 proposed patches across seeds, 14 touched a protected path and were rejected by the path guard, 9 were rejected by the static safety linter (all nine added network access or a shell escalation

that the task's policy disallowed, and none was an attempt to reach the verifier), 15 failed compilation, unit, or regression tests (9 at compilation or unit tests and 6 at the regression suite), and 195 were committed. Of the 233 proposals, 201 reached the regression stage, an average of one evaluation per round. The leakage scan, which searches every diff for task identifiers, expected-answer literals, and benchmark-specific paths, rejected no committed patch in the main runs. In a pilot without the anti-specialisation guidance it flagged 7 of 31 patches, so both the guidance and the scan are needed and the scan is the enforceable one.

Two of the authors independently read 40 committed diffs sampled uniformly across rounds and seeds, with access to the surrounding source. They rated 34 as changes a maintainer would accept, 6 as functional but in need of refactoring (mostly duplicated helper logic across tools), and none as unsafe. This is a small audit by non-blind raters and is reported as a description.

The linter and sandbox address privilege escalation, network policy, and secret access at the level of static patterns and container policy. They do not test for reward hacking through verifier-adjacent behaviour that stays within the allowed surface, for adversarial tool outputs, or for behaviours that appear only under distribution shift. A co-evolution loop that edits its own runtime warrants runtime monitoring as well as gating, and we regard the audit here as the minimum a paper in this setting should report.

U Case Studies

The six cases show how allocation decisions affect individual tasks. Success estimates use the eight measurement rollouts per state, and the change scores are the null-corrected values of Eq. 5 with e_0 evaluated at $N = 8$. When a task also appears in a training group under the same state, its measurement pools those rollouts (Appendix D), giving some success rates finer resolution than $1/8$. Costs are generated tokens summed over the turns of a trajectory, and the step limits are listed in Appendix H. Traces and diffs are abridged, with elisions marked by ellipses or bracketed notes. *Self-Judgment* outputs appear as log records reformatted into fields, where the label is the leading word of the raw response and the reason is the sentence that follows.

U.1 Case 1. A harness edit creates work for the model

Task.

```
The project in /app fails to build. Run ‘make‘ to reproduce the failure,
find the cause, and fix it so that ‘make clean && make‘ completes and
produces ./bin/imgproc. Do not modify tests/ or the test script.
```

The Makefile sets `MAKEFLAGS += -k`, so the build continues after the failure and emits 1,291 lines. The single line that matters is line 43.

```
src/filters.c:12:10: fatal error: imgcore/kernel.h: No such file or directory
```

At round 10 the task succeeds on two rollouts in eight. The failed trajectories share one failure mode, shown here from rollout 3.

```
[step 07] shell{cmd: "make 2>&1"}
  -- output truncated: showing last 200 of 1291 lines --
  src/render.c:88:12: warning: unused variable 'tmp' [-Wunused-variable]
  src/render.c:91:5:  warning: implicit declaration of 'clampf'
  cc  -o bin/imgproc build/main.o build/render.o build/filters.o
  cc: error: build/filters.o: No such file or directory
  make: *** [Makefile:34: bin/imgproc] Error 1
[step 08] assistant: The link step cannot find filters.o. I will inspect the
          Makefile rule for that object.
[step 09] shell{cmd: "sed -n '20,40p' Makefile"}
[step 12] shell{cmd: "make 2>&1 | tail -n 60"}
  (the last 60 lines, ending at the link failure above)
[step 15] shell{cmd: "make 2>&1"}
  -- output truncated: showing last 200 of 1284 lines --
```

```
[steps 16--29] the same commands with minor variations
[step 30] -- step budget exhausted --
```

The agent never sees line 43. The tool keeps only the last 200 lines of any result, and narrowing the command with `tail` shows even less of the log. Rollouts $1, 2, 4, 5, 6$ repeat this with minor variations, rollouts 7 and 8 succeed after guessing the include path.

At round 11 these trajectories enter $\mathcal{T}^{\text{harness}}$ and the evolver commits.

```
--- a/harness/tools/shell.py
+++ b/harness/tools/shell.py
@@ class ShellTool:
-    MAX_CAPTURE_LINES = 200
+    MAX_CAPTURE_LINES = 2000
+    _last_log = ""

     def run(self, cmd, timeout=120):
         proc = subprocess.run(cmd, shell=True, capture_output=True,
                               timeout=timeout, text=True)
-        return tail(proc.stdout + proc.stderr, self.MAX_CAPTURE_LINES)
+        full = proc.stdout + proc.stderr
+        ShellTool._last_log = full
+        out = tail(full, self.MAX_CAPTURE_LINES)
+        if full.count("\n") > self.MAX_CAPTURE_LINES:
+            out += ("\n-- %d earlier lines retained; search them with "
+                    "build_log_grep --" % (full.count("\n")
+                                           - self.MAX_CAPTURE_LINES))
+        return out

--- /dev/null
+++ b/harness/tools/build_log_grep.py
+def build_log_grep(pattern: str, context: int = 2) -> str:
+    """Search the full output of the most recent shell call.
+
+    Use this when a build or test command produced more output than the
+    tool result showed.  Anchor the pattern, for example
+    '^[^ ]+:[0-9]+:[0-9]+: (fatal )?error:' matches compiler errors and
+    fatal errors and leaves out warnings.
+    """
+    log = ShellTool._last_log
+    if not log:
+        return "no previous shell output retained"
+    return grep_with_context(log, pattern, context)[:8000]

--- a/harness/manifest.yaml
+++ b/harness/manifest.yaml
   - shell(cmd, timeout)
+  - build_log_grep(pattern, context=2)
```

Measuring $(\theta_{11}, H_{10}) \to (\theta_{11}, H_{11})$ with the pooled eight rollouts per state gives $\Delta = \max\{0, |5/8 - 2/8| - 0.195\} + 0.2 \cdot \frac{2}{3} \cdot |29.1 - 40.9|/43.6 = 0.180 + 0.036 = 0.216$, where $e_0(0.44) = 0.195$ is the null magnitude at the pooled success level, $c_{\text{ref}}(x) = 43.6$k tokens is the task's round-0 successful cost, and $\rho_c = 2/3$ because the earlier state has two successes. This is in the top 3% of that round's distribution, and the task enters $\mathcal{T}^{\text{model}}$.

What follows is the part an aggregate curve hides. The policy does not use the new tool well at first.

```
round 12, rollout 2:
[step 04] build_log_grep{pattern: "error", context: 2}
  -> 340 matching lines, first 8,000 characters returned (2.4k tokens)
[steps 05--15] build_log_grep with broader patterns, each returning 8,000 characters
[step 16] -- context window exceeded, oldest interactions truncated --
[steps 17--29] further broad patterns, each result again 8,000 characters
[step 30] -- step budget exhausted --
```

At round 12 the success change of one rollout in eight lies below the null magnitude, so the corrected change score falls to 0.014 and the task stays in the model batch through its *Self-Judgment* label, which attributes the overflowing greps to the model. By round 14, after training on exactly these trajectories, the policy stops issuing broad greps and works from the enlarged build output directly.

```
round 14, rollout 1:
[step 03] shell{cmd: "make 2>&1"}
  -- 1291 lines, no truncation --
  src/filters.c:12:10: fatal error: imgcore/kernel.h: No such file or dir
[step 04] shell{cmd: "grep -n 'CFLAGS' Makefile"}
  8: CFLAGS = -Wall -O2 -Iinclude
[step 05] shell{cmd: "ls third_party/imgcore/include"}
  imgcore/
[step 06] shell{cmd: "sed -i \
  '8s|-Iinclude|-Iinclude -Ithird_party/imgcore/include|' \
  Makefile"}
[step 07] shell{cmd: "make clean && make"}
  cc -o bin/imgproc ...   [OK]
```

Round	p	cost (k)	Δ	routed to	event
10	0.25	40.9	—	—	first error truncated from build output
11	0.625	29.1	0.216	model (by Δ)	window widened, `build_log_grep` added
12	0.50	32.8	0.014	model (by b)	over-broad grep patterns overflow context
14	0.875	19.7	0.088	—	policy drops broad greps, reads full output

The harness edit alone is worth $+0.375$, training the model on the same task because of that edit is worth a further $+0.25$ and halves the token cost. An allocation rule that does not measure the counterpart's effect would have sampled this task at its base rate in round 11, and the second half of the gain would have arrived slowly or not at all.

U.2 Case 2. A model update makes part of the harness obsolete

Task.

```
Q: Which magazine was started first, Arthur's Magazine or First for Women?

Gold: Arthur's Magazine
```

The Search-o1 initial harness applies the same three-stage retrieval pipeline to every sub-question. After extracting an answer from the first retrieval, it unconditionally performs a second search and verification step.

```
# harness/flow/subquestion.py   (H_18)
def answer_subquestion(q):
    hits = search(q, k=5)
    span = extract(q, hits)
    check = search(f"{span} {q}", k=3)      # unconditional second retrieval
    if not verify(span, check):
        return retry(q)
    return span
```

By round 19, the updated policy can often establish the requested fact directly from the first retrieval. This changes how the same harness is used, even when the final task outcome does not change. One of the trajectories routed to the harness evolver contains the following sub-question.

```
state: (\theta_19, H_18)

[step 03] search{q: "Arthur's Magazine founded", k: 5}
  [1] "Arthur's Magazine (1844-1846) was an American literary periodical
       first published in Philadelphia in 1844."
```

```
[step 04] extract{q: "Arthur's Magazine founded", hits: [...]}
          -> "1844"

[step 05] assistant:
          The retrieved sentence explicitly states that Arthur's Magazine
          was first published in 1844.

[step 06] search{q: "1844 Arthur's Magazine founded", k: 3}
  [1] "Arthur's Magazine was first published in Philadelphia in 1844."

[step 07] verify{span: "1844", check: [...]} -> supported
```

The second retrieval does not supply new evidence in this trajectory. The answer span is already explicitly supported by the sentence from which it was extracted, but the fixed control flow still executes the additional search and verification stages.

The model-induced comparison $(\theta_{18}, H_{18}) \to (\theta_{19}, H_{18})$ makes this task informative for the subsequent harness update. With the eight measurement rollouts used by the main experiment, both states succeed on all eight rollouts, while mean successful cost decreases from 24.0k to 18.8k generated tokens. Thus $p^- = p^+ = 1$, the null magnitude is $e_0(1) = 0$, the reliability factor is $\rho_c = 8/(8+1) \approx 0.89$, and with $c_{\text{ref}} = 26.0$k,

$$\Delta = \max\{0,\, |1-1| - 0\} + 0.2 \cdot 0.89 \cdot \frac{|18.8 - 24.0|}{26.0} = 0.036.$$

Several other tasks in the current candidate panel exhibit the same pattern, where the first retrieval already contains the evidence used for the answer but the harness nevertheless performs the unconditional verification stage. Five such tasks enter the sixteen positions of $\mathrm{TopK}(w_{\text{harness}})$ in this round.

Across the routed examples, the evolver can directly inspect that the extracted answer is already supported by the retrieval sentence before the second search is issued. It therefore replaces unconditional verification with a deterministic evidence-presence check.

```
--- a/harness/flow/subquestion.py
+++ b/harness/flow/subquestion.py
 def answer_subquestion(q):
     hits = search(q, k=5)
-    span = extract(q, hits)
+    span, evidence = extract_with_evidence(q, hits)
+
+    # Skip the second retrieval only when the extracted answer is
+    # explicitly supported by the sentence from which it was taken.
+    if directly_supports(q, span, evidence):
+        return span
+
     check = search(f"{span} {q}", k=3)
     if not verify(span, check):
         return retry(q)
     return span

--- a/harness/manifest.yaml
+++ b/harness/manifest.yaml
-  - extract(q, hits) -> span
+  - extract_with_evidence(q, hits) -> (span, evidence)
+       returns the extracted span together with the retrieval sentence
+       from which the span was obtained
+  - directly_supports(q, span, evidence) -> bool
+       returns true only when the evidence explicitly states the
+       information required by the sub-question
```

The edit is not tied to this question, either magazine name, or the expected answer. It introduces a general fast path for sub-questions whose first retrieval already contains explicit evidence, while preserving the original second-search and retry path whenever direct support is absent.

Under the updated harness H_{19}, the same task executes as follows.

```
state: (\theta_19, H_19)

[step 03] search{q: "Arthur's Magazine founded", k: 5}
  [1] "Arthur's Magazine (1844-1846) was an American literary periodical
       first published in Philadelphia in 1844."

[step 04] extract_with_evidence{q: ..., hits: [...]}
          -> span: "1844"
             evidence: "Arthur's Magazine ... first published ... in 1844."

[step 05] directly_supports{
              q: "Arthur's Magazine founded",
              span: "1844",
              evidence: "Arthur's Magazine ... first published ... in 1844."
          }
          -> true

[step 06] return "1844"
```

The corresponding sub-question for *First for Women* is resolved in the same way from a retrieved sentence giving its start year, after which the agent compares the two dates and returns *Arthur's Magazine*.

Across the affected tasks, the harness edit reduces mean successful execution cost because directly supported sub-questions no longer invoke the second retrieval. Tasks for which the first retrieval is ambiguous continue through the original verification path, so the edit removes redundant execution without removing the fallback mechanism that handles weak evidence.

This case illustrates why Behavioral-Change can provide useful harness-update signal even when a task is already solved successfully. The relevant change is not necessarily a transition from failure to success. Here the updated policy uses the existing retrieval interface differently, making part of the old control flow redundant. A routing rule based only on failures would not prioritize such a task, whereas Δ can assign it nonzero weight through the change in successful execution cost.

U.3 Case 3. Self-Judgment recovers missed tasks

```
Q: In the attached 2019 annual report, what was the reported R&D expenditure
   in the year with the highest total revenue?  Give the figure in millions,
   rounded to the nearest whole number.
   Attachment: report.pdf (34 pages)
Gold: 412
```

The revenue series and the R&D series appear only as bars in Figure 4 on page 12, the surrounding text names neither value. Every measured rollout from round 0 to 22 ends the same way.

```
[step 02] pdf_extract_text{page: 12}
  -> "\n\n\n\nFigure 4: Revenue and R&D expenditure by fiscal year\n\n\n
      Source: audited consolidated statements.\n\n"
[step 03] pdf_extract_text{page: 12, mode: "raw"}
  -> ""
[step 04] pdf_search{query: "R&D expenditure"}
  -> p.12 "Figure 4: Revenue and R&D expenditure by fiscal year"
     p.31 "Research and development is expensed as incurred."
[step 05] assistant: The figure is an image; the extractor returns no text for
          that region and there is no tool that renders the page or reads the
          axis labels.  I cannot recover the values.
```

$p = 0$ at every measurement, so $\Delta^{\mathrm{M}} = \Delta^{\mathrm{H}} = 0$ exactly. Under a purely multiplicative weight, $w_a(x) = \Delta^{\bar{a}}(x)\,(1 + \beta_t \mathbb{1}[\cdot])$, these tasks carry zero weight in every round and are never routed to either side, however confidently they are attributed. The *Self-Judgment* call returns the same thing 22 rounds running.

```
{"self_judgment": "harness",
 "reason": "pdf_extract_text returns an empty string for the region containing
            Figure 4 and pdf_search only reaches the caption. No tool renders a
            page as an image or reads axis labels, so the plotted values are
            unreachable with the current toolset."}
```

At round 23, $\beta_{23} = 0.21$ while the median non-zero Δ that round is 0.10. The additive term alone gives $w_{\text{harness}} = 0.21$, above the median, and five of the seven tasks enter TopK. The evolver commits.

```
--- /dev/null
+++ b/harness/tools/pdf_render.py
+def render_pdf_page(page: int, dpi: int = 200, bbox=None) -> Image:
+    """Rasterise one page (or a region of it) so that it can be passed to
+    describe_image().  Use when pdf_extract_text returns empty or returns
+    only a caption for a page that visibly contains a figure."""
+    pg = _doc.load_page(page - 1)
+    pix = pg.get_pixmap(dpi=dpi, clip=bbox)
+    return pix.to_pil()

--- a/harness/manifest.yaml
+++ b/harness/manifest.yaml
   - pdf_extract_text(page, mode="layout")
   - pdf_search(query)
+  - render_pdf_page(page, dpi=200, bbox=None) -> image
+       pair with describe_image() to read figures; pdf_extract_text
+       returning "" for a page is the signal to use this
```

The recovered trajectory at round 24. The `describe_image` tool sends the rendered page to the task model itself through its image input.

```
[step 02] pdf_extract_text{page: 12}   -> caption only
[step 03] render_pdf_page{page: 12, dpi: 200}   -> <image 1700x2200>
[step 04] describe_image{image: ..., q: "read every bar label in Figure 4"}
  -> "Grouped bars, FY2015-FY2019. Revenue (dark): 3120, 3400, 3610, 3980,
      4270. R&D (light): 305, 331, 358, 389, 412. Values in $M."
[step 05] assistant: Highest revenue is FY2019 at 4270, so R&D is 412.
```

Round	p	Δ^{H}	Δ^{M}	β_t	w_{model}	w_{harness}	routing outcome
0–22	0.00	0.00	0.00	0.00–0.20	0	β_t	below TopK
23	0.00	0.00	0.00	0.21	0	0.21	enters TopK, renderer added
24	0.50	0.33	0.00	0.22	0.55	0	enters the model batch
30	0.63	0.00	0.06	0.22	0.22	0.06	—

The renderer added by the harness edit at round 23 produces $\Delta^{\text{H}} = 0.33$ at round 24, and the counterpart-aware rule routes that change to the model, whose failures on these tasks are now attributed to the model.

In the $\beta \equiv 0$ ablation these seven tasks are never routed and remain at $p = 0$ through round 40. This is the case the additive form exists for. A task that nothing has moved carries no information in Δ, and a persistent failure that no update has disturbed is precisely the kind a co-evolution loop is otherwise blind to.

U.4 Case 4. Adaptive weighting absorbs an incorrect attribution

In this case, the task's own verification script contains a race that the idle-container determinism screen did not expose, and the script is not editable by the agent.

```
# tests/test_outputs.sh  -- part of the task definition, read-only
/app/bin/exporter --config /app/exporter.yml &
sleep 0.2
curl -sf http://localhost:9100/metrics | grep -q 'exporter_up 1'
```

Under load the exporter binds its socket later than 200 ms after launch, so roughly one rollout in six fails on an otherwise correct solution. The agent's *Self-Judgment* call.

```
{"self_judgment": "harness",
 "reason": "the same solution passed on the previous attempt and failed here
            with no difference in my actions; a retry around the verification
            step would make the outcome deterministic."}
```

The reasoning is locally sound and the conclusion is wrong. At round 9 the evolver acts on it.

```
--- a/harness/flow/verify.py
+++ b/harness/flow/verify.py
 def run_verification(cmd):
-    return sh(cmd)
+    last = None
+    for attempt in range(3):          # added round 9
+        last = sh(cmd)
+        if last.ok:
+            return last
+    return last
```

The edit does not help in the measurements. The intervention gain is $G = -0.03$, with success unchanged within noise and token cost up 7%, so the prediction is scored incorrect. Each retry re-executes the task's own script, which relaunches the exporter under the same load, and the unchanged success rate is consistent with the late binding recurring across attempts. Each attempt is charged to the budget.

Three further tasks in this cluster behave identically. Together they move measured *Self-Judgment* intervention success rate from 0.72 at round 8 to 0.64 at round 12, and β_t from 0.22 to 0.14. That reduction is indiscriminate, it lowers the influence of every correct attribution as well, but it also caps what the evolver spends on unrepairable failures. By round 13 the cluster's weight has fallen below TopK and is not revisited, and intervention success rate recovers to 0.69 by round 16.

The mechanism never identifies which attribution was wrong. It reduces the weight of the whole channel until the channel's predictions begin validating again. That is a coarse correction, and it is the reason β is measured.

U.5 Case 5. Predicted responsibility shifts within a task family

In this case the predicted attribution moves within a single task family, and the calibration set lets us check it against matched interventions. Task.

```
Install the service from /app/dist/edge-agent-2.4.1.tar.gz, configure it to
listen on port 8443 with TLS using the certificate and key in /app/certs,
and leave it running.  systemctl is not available in this container.
```

Rounds 0–6, attributed to the harness (7 of 8). The matched interventions of Appendix N agree. The shell tool waits for the child to exit, so launching a daemon consumes the whole tool timeout.

```
[step 06] shell{cmd: "/opt/edge-agent/bin/edge-agent --config /etc/edge.yml"}
  -> (no output; tool timed out after 120s)
[steps 07--29] the same launch with variations, each timing out
[step 30] -- step budget exhausted --
```

Round 7 edit.

```
--- /dev/null
+++ b/harness/tools/proc.py
+import socket
+import subprocess
+import time
+
```

```
+def shell_bg(cmd: str) -> int:
+    """Start a command in the background and return its pid immediately.
+    Use for daemons; the blocking shell() will hold until timeout."""
+    p = subprocess.Popen(cmd, shell=True, start_new_session=True)
+    return p.pid
+
+def wait_for_port(port: int, timeout: int = 30) -> bool:
+    """Block until a TCP listener appears on port, or timeout."""
+    deadline = time.monotonic() + timeout
+    while time.monotonic() < deadline:
+        with socket.socket() as s:
+            s.settimeout(0.5)
+            if s.connect_ex(("127.0.0.1", port)) == 0:
+                return True
+        time.sleep(0.25)
+    return False
```

Rounds 7–15, attributed to the model (5 of 8). The matched interventions again agree. The tools now exist and the remaining failures are ordering.

```
[step 05] shell_bg{cmd: "/opt/edge-agent/bin/edge-agent"}      -> pid 214
[step 06] shell{cmd: "cat > /etc/edge-agent.yml <<'EOF' ... EOF"}
[step 07] wait_for_port{port: 8443, timeout: 30}               -> False
[step 08] shell{cmd: "ss -lntp | grep edge"}
  LISTEN 0 128 0.0.0.0:8080  users:(("edge-agent",pid=214,fd=7))
```

The daemon read the default configuration at startup and bound 8080. Reinforcement learning over rounds 8–14 fixes the ordering, and family success rises from 0.25 to 0.63.

Rounds 16–40, split. The residual failures are genuine edge cases, one task whose configuration schema is undocumented in the container, and the attribution settles into a roughly even split that does not resolve further. Across the family the harness-*Self-Judgment* share runs 88%, then 38%, then 50%, which is a step down once the harness deficit is repaired and a partial rebound once a new mechanism has to be learned.

U.6 Case 6. The harness sheds an obsolete component

In this case, a harness optimisation that was correct for the round-0 policy becomes actively harmful once the policy improves, and the evolver removes it.

The instance requires editing one of four near-identical helpers in the same file.

```
Repository: pyfmt
Problem:  Three of the four _normalise_* helpers strip trailing whitespace
          from the first line.  _normalise_block does not, so block comments
          keep a trailing space and round-trip formatting is unstable.  Make
          the four consistent without changing _normalise_inline's behaviour
          on escaped newlines.
FAIL_TO_PASS: tests/test_normalise.py::test_block_roundtrip
PASS_TO_PASS: tests/test_normalise.py  (14 tests)
```

The round-0 harness keeps tool-call payloads small by truncating the search string before matching, and resolves the resulting ambiguity with a nearest-context heuristic.

```
# harness/tools/edit.py   (H_0)
MAX_SEARCH_LINES = 3        # keep tool-call payloads small

def apply_edit(path, search, replace):
    key  = "\n".join(search.splitlines()[:MAX_SEARCH_LINES])
    hits = find_all(read(path), key)
    if not hits:
        return "search string not found"
    if len(hits) > 1:
```

```
        hits.sort(key=lambda h: context_distance(h, search))
        return commit(path, hits[0], replace) + \
               " (%d candidates; chose the closest)" % len(hits)
    return commit(path, hits[0], replace)
```

Early in the run this is the right trade. The untrained policy emits short, often mis-indented search strings, the truncation makes them match anyway, and the heuristic picks correctly on 81% of the ambiguous calls it resolves.

```
round 8, rollout 2:
[step 11] apply_edit{path: "pyfmt/normalise.py",
                     search: "    if not line:\n        return line",
                     replace: "    if not line:\n        return line.rstrip()"}
  -> applied (3 candidates; chose the closest)
```

By round 21 the policy has learned to quote long, unambiguous context, mean search length rises from 2.6 to 8.2 lines over rounds 12–20, and the truncation now discards exactly the lines that carried the disambiguation.

```
round 21, rollout 1:
[step 09] apply_edit{path: "pyfmt/normalise.py",
                     search: "    lines = src.splitlines()\n"
                             "    if not lines:\n"
                             "        return src\n"
                             "    head = lines[0]\n"
                             "    body = lines[1:]\n"
                             "    if not body:\n"
                             "        return head\n"
                             "    out = [head]",
                     replace: "...  head = lines[0].rstrip()  ..."}
  -> applied (4 candidates; chose the closest)
[step 10] shell{cmd: "git diff --stat"}
  pyfmt/normalise.py | 4 ++--
[step 11] shell{cmd: "sed -n '61,69p' pyfmt/normalise.py"}
  def _normalise_inline(src: str) -> str:      <-- the edit landed here
      lines = src.splitlines()
[step 12] shell{cmd: "python -m pytest tests/test_normalise.py -x"}
  FAILED test_inline_escapes - AssertionError
  FAILED test_block_roundtrip - AssertionError
```

The policy supplied eight lines, and the fourth is the first that separates `_normalise_block` from the other helpers. The harness matched on the first three, found four candidates, and silently chose `_normalise_inline`. The edit also breaks a `PASS_TO_PASS` test, so the instance fails in a way it did not fail at round 12.

Measuring $(\theta_{20}, H_{20}) \to (\theta_{21}, H_{20})$ over the 18-instance cluster whose target file contains repeated near-identical blocks pools eight rollouts per instance and state, giving 63 and 45 successes among 144 rollouts. Routing scores are computed per instance. One instance has four successes among its eight rollouts at the earlier state and one at the later, so $\rho_c = 1/2$ and $\Delta = \max\{0, |1/8 - 4/8| - 0.181\} + 0.2 \cdot \frac{1}{2} \cdot |29.8 - 33.7|/35.2 = 0.194 + 0.011 = 0.205$ with $e_0(0.31) = 0.181$ and $c_{\mathrm{ref}} = 35.2$k tokens, and eleven of the 18 instances have a nonzero corrected score. The success change is negative. The model update made this cluster worse. Aggregate SWE-bench Pro success rose from 51.2 to 52.4 in the same round, so the regression is invisible in the headline number and visible only per task.

The evolver receives the cluster together with the current source, observes that the mean supplied search length now exceeds `MAX_SEARCH_LINES` by a factor of nearly three, and deletes the optimisation.

```
--- a/harness/tools/edit.py
+++ b/harness/tools/edit.py
-MAX_SEARCH_LINES = 3        # keep tool-call payloads small
-
```

```
 def apply_edit(path, search, replace):
-    key  = "\n".join(search.splitlines()[:MAX_SEARCH_LINES])
-    hits = find_all(read(path), key)
+    hits = find_all(read(path), search)
     if not hits:
-        return "search string not found"
+        return ("search string not found verbatim; use read_file to copy "
+                "the exact lines, including indentation")
     if len(hits) > 1:
-        hits.sort(key=lambda h: context_distance(h, search))
-        return commit(path, hits[0], replace) + \
-               " (%d candidates; chose the closest)" % len(hits)
+        return ("search string is not unique: %d matches, at lines %s. "
+                "Extend it until it matches once."
+                % (len(hits), [h.line for h in hits]))
     return commit(path, hits[0], replace)

-def context_distance(hit, search):
-    """Rank candidate hits by similarity of surrounding context."""
-    ...                                          # 31 lines removed
-
--- a/harness/manifest.yaml
+++ b/harness/manifest.yaml
-  - apply_edit(path, search, replace)
-      search is matched on its first 3 lines; if several places match,
-      the closest by surrounding context is chosen
+  - apply_edit(path, search, replace)
+      search must match verbatim and exactly once; on multiple matches
+      the call fails and reports the candidate line numbers
```

The committed patch removes 41 lines and adds 9. At round 22 the same instance succeeds on six rollouts of eight and the cluster success rate rises to 0.67 (96 of 144 rollouts). Where the search string is ambiguous the tool now says so.

```
round 22, rollout 1:
[step 09] apply_edit{path: "pyfmt/normalise.py", search: "    lines = ...",
                     replace: "..."}
  -> search string is not unique: 4 matches, at lines 13, 35, 62, 89.
     Extend it until it matches once.
[step 10] apply_edit{path: "pyfmt/normalise.py",
                     search: "def _normalise_block(src: str) -> str:\n..."}
  -> applied
```

Round	p	cost (k)	Δ	routed to	event
8	0.44	34.9	—	—	truncation helps; heuristic right on 81%
20	0.44	33.7	—	—	mean search length $2.6 \to 8.2$ lines
21	0.31	29.8	0.205	harness	truncation discards disambiguation
22	0.67	27.4	—	—	optimisation deleted, 41 lines removed

We can see from this case that, first, the component was never defective, it was correct for the policy that existed when it was written, and the policy is what changed. Second, its value inverted in sign and did not simply decay to zero, and it passed through zero somewhere around round 16, which is precisely when an acceptance gate keyed on immediate measured gain, such as HASE's, has no reason to touch it in either direction. Third, no failure-keyed allocation rule surfaces this, since at round 20 the cluster's success rate matched its round-8 level and at round 21 the additional failures look like ordinary task failures and carry no visible mark of the harness. What identified the component was that the model's own improvement changed the shape of its calls.

Deletion is a kind of edit an evolver rarely proposes unprompted. The instruction in Appendix H that removals count as edits, and that a mechanism which no longer appears in successful trajectories should be narrowed or removed, is what makes this patch reachable. In the Terminal-Bench 2 runs, 6 of the 39 committed edits per run, averaged over five seeds, are net deletions, and total harness

length peaks at round 26 and ends 6% below that peak, which means the harness grows for most of the run and then begins, slowly, to get smaller (Appendix T tracks this over rounds).

V SENSITIVITY TO λ AND K

Table 26: Round-40 held-out success on Terminal-Bench 2 as the cost-term weight λ and the evolver batch size K are varied around their defaults, $\lambda = 0.2$ and $K = 16$. Two seeds per varied setting, with the default entries repeating the five-seed value of Table 1.

λ	0.0	0.1	0.2	0.4	0.8
round-40 held-out	43.1	44.2	44.6	43.8	41.6
K	4	8	16	32	64
round-40 held-out	41.9	43.7	44.6	44.1	42.4

Both hyperparameters have interior optima and shallow neighbourhoods (Table 26). Setting $\lambda = 0$ discards the cost term and costs 1.5 points, which is the size of the effect in Case 2. Without a cost signal, an update that changes how a task is solved without changing whether it succeeds produces $\Delta = 0$ and is never routed. Raising λ to 0.8 costs 3.0 points by letting token-count fluctuations outweigh success changes. K trades coverage against dilution in the evolver's context. At $K = 4$ the evolver sees too few trajectories to distinguish a recurring deficit from a single unlucky rollout, and at $K = 64$ the context spans unrelated failures and the resulting edits address several at once, which raises the regression gate's rejection rate. Because $K = 64$ exceeds the 44-task Terminal-Bench 2 panel, the panel is enlarged to 128 tasks at that setting, its measurements are charged to the same per-round cap, and the realized number of GRPO iterations falls from 15 to 14. The other settings use the default panel. Across the tested settings the final score remains within 3.0 points for λ and 2.7 points for K, and with two seeds these differences are descriptive.

W LIMITATIONS

The alternating model–harness loop, the harness evolver, the regression gate, and the GRPO update are shared with prior systems, and our contribution is the allocation step and its validation. The verifier, task pools, update operators, editable surface, and budget rules are fixed by us and are not modified by the system, so "recursive self-improvement" is used here in the narrow sense of alternating self-modification of two components under a fixed outer loop, not in the open-ended sense.

The counterfactual labels come from two small single-task interventions with $N = 4$, so they are noisy, and 14–19% of calibration tasks are ties. Human labels agree with them 71% of the time and with each other moderately. Calibrated accuracy is therefore an estimate against an imperfect reference, and the 16-task calibration set per round is small.

Across the four judges tested, performance is monotone in the judge's calibrated accuracy. The frozen round-0 judge improves by 8 points of calibrated accuracy against 16 for the evolving self-judge, and because the two conditions evolve separate runs this comparison does not isolate the effect of the task model becoming stronger. The main configuration also uses the task model as the harness evolver. We varied the judge but not the evolver, so the interaction between the allocation rule and a systematically different evolver is uncharacterised.

The main tables use $N = 8$ and the null-corrected score, fixed from a pilot on the Terminal-Bench evolution pool. The comparison of $N \in \{4, 8, 16, 32\}$ and of the corrected and raw scores was run on Terminal-Bench 2 with three seeds, and $N = 16$ scores 0.3 points higher than $N = 8$ under the matched cap. At $N = 8$ a change of one rollout in eight lies below the null magnitude and receives no success-change weight, so late-round routing depends increasingly on attribution.

On the strong backbone, the oracle-replay time schedule is within 0.8 points of the adaptive rule and not distinguishable from it at the task level. The adaptive rule's observed advantage is that it finds this profile without tuning and adjusts it when the judge is unreliable. It is not a large advantage over a well-chosen ramp on a strong model.

The main horizon is 40 rounds and the 80-round extension uses one benchmark and two seeds. GAIA evolves on 65 tasks. The private test set preserves the held-out ordering, although its unpaired interval includes zero, and a larger disjoint evolution pool, as used for Terminal-Bench 2, would be preferable.

Standardized adaptations differ from the published systems. Appendix E documents each change and reports native runs, and the HELIX portfolio is budget-limited to 8 candidates. HarnessForge's SFT alignment is reported natively but the standardized row uses GRPO.

The design is expensive, and the ledger makes the cost explicit. The appendix studies use three seeds, and two for some sweeps, which supports descriptive comparisons only. Seed intervals are two to three times narrower than task intervals on the small held-out sets, and the schedule comparison is an example of a difference that the seed intervals suggest and the task intervals do not support.

The audit is static and small. The linter and sandbox cover privilege, network, and secret access at the level of patterns and container policy. They do not cover reward hacking within the allowed surface or behaviour under distribution shift, and the harness becomes larger and more complex over the run.