GYMHARNESS: RECURSIVE SELF-IMPROVEMENT OF AGENTS WITH RECURSIVELY VERIFIED VERIFIERS

Anonymous authors
Paper under double-blind review

ABSTRACT

A self-improving agent learns from data it generates itself, so how far it improves is governed by the verifier that labels that data. When ground truth is scarce, the verifier must be built by the system, and a fixed verifier does not stay accurate: as the policy trains, it leaves the distribution on which the verifier was calibrated, and the verifier accepts more of its errors. We present GymHarness, a recursive self-improvement framework in which a policy and a smaller verifier improve each other through supervised fine-tuning, and in which every verifier update is itself verified. A candidate verifier replaces the current one only if it passes a hidden, append-only probe suite whose labels come from execution evidence that the agent cannot write, so the recursion bottoms out in exogenous evidence rather than in the model's own judgment. Trained on its own verifier-filtered rollouts, with golden-derived hints only on near-zero-pass tasks, Qwen3.6-27B improves by 11.3 points on WebShop, ALFWorld, and DBBench and by 5.2 points on two held-out environments, recovering 89% of the gain of oracle-filtered training with fixed anchors and a 10% adaptive rollout-audit rate. A 9B verifier suffices to supervise the 27B policy, and co-evolving the verifier without the recursive gate performs worse than not evolving it at all.

1 INTRODUCTION

Recursive self-improvement (RSI) asks a system to turn its own experience into lasting improvements. For language agents it currently takes two forms. Harness-level RSI keeps the model fixed and evolves the prompts, tools, memory, and control flow around it; iterative harness evolution has become a practical form of RSI at the agent-system level (Liu et al., 2026; Shao et al., 2026; Wu et al., 2026a; Xia et al., 2026). Self-training instead updates the model on self-generated trajectories that pass a correctness filter (Gulcehre et al., 2023; Singh et al., 2023; Zelikman et al., 2022). Both loops rest on a component that neither of them improves: the verifier that decides which experience counts as success.

That verifier is usually treated as a fixed, trustworthy oracle such as an answer key, a unit test, or a benchmark evaluator. The assumption fails precisely where self-improvement is meant to operate. Ground truth is scarce for self-generated experience, and agents that build their own signals train on mismatched data and over-trust narrow self-evaluations (Wu et al., 2026b); in data-centric self-improvement, 78.26% of searches that continue past their best score end lower (Meng et al., 2026). Checks are exploitable: 16% of 1,968 audited terminal-agent tasks are hackable from the task description alone (Zhong et al., 2026), and a self-improving coding agent has proposed patches that relax its own evaluation harness (Fu et al., 2026). And checks go stale: no fixed reward remains effective as the policy grows (Wang et al., 2026a), and static rubrics lose their discriminative power as the model improves (Ding et al., 2026).

Figure 1 shows the outcome. When Qwen3.6-27B trains on its own trajectories filtered by a frozen Qwen3.5-9B verifier, the verifier's precision on the data it accepts falls from 0.86 to 0.71 over four rounds, and the policy stalls at 67.0%: it increasingly learns what the verifier accepts instead of the task. Retraining the verifier without an external gate preserves some negative-class rejection but loses completeness on alternative-correct solutions and yields an even weaker policy (65.0%). Self-improvement needs a verifier that improves with the policy, while an evidence-backed gate protects both soundness and coverage of legitimate solutions.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

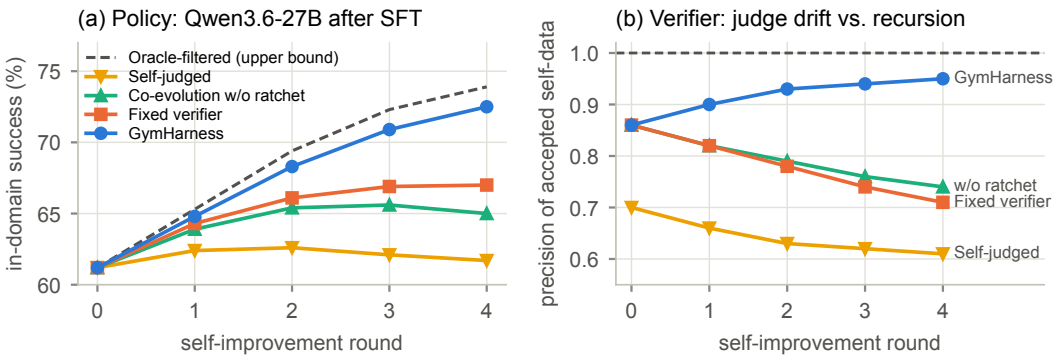

Figure 1: Self-improvement is bounded by its verifier. Qwen3.6-27B is fine-tuned for four rounds on its own trajectories, with fixed anchors and a 10% adaptive rollout-audit rate. (a) Sealed success of each round's checkpoint, evaluated offline after all selection decisions were final; no sealed result reaches training or selection. (b) Precision of the verifier on the self-generated data it accepts. A frozen verifier drifts as the policy improves, and a verifier co-evolved without a recursive gate drifts as well; GymHarness keeps precision rising and approaches the oracle-filtered upper bound.

We introduce **GymHarness**, an RSI framework built around a *recursive verifier*. A smaller generative verifier labels the policy's trajectories, routes their failures, and filters the data on which the policy is fine-tuned; the new policy's behavior then becomes training data for the next verifier. The recursion rests on how updates are accepted. A policy update is kept only when a paired validation test supports it. A verifier update is kept only when it passes a hidden, append-only probe suite of golden and alternative-correct trajectories that must pass and null, perturbed, and hacked trajectories that must fail, all labeled by sandboxed execution or anchor checks. Because this suite contains no learned judgment, the recursion bottoms out in exogenous evidence. Three principles shape the design: evidence remains outside the agent's write surface, every recursion has a base case, and every update is atomic and reversible.

GymHarness audits at most 10% of each round's rollouts against the success check. This rate excludes the fixed anchors, validation checks, probe execution, and hint-derived trajectories that also supply executable supervision: in total the loop makes 274 execution calls per 100 rollout episodes, 10 of them audits (Appendix G.2). With this supervision, GymHarness raises Qwen3.6-27B from 61.2% to 72.5% on WebShop, ALFWorld, and DBBench, and from 39.0% to 44.2% on two environments it never trains on, recovering 89% of the improvement of oracle-filtered training, which labels every rollout. It exceeds training with a frozen verifier under the same evidence by 5.5 points and distillation from a much larger teacher by 1.6 points, and a 9B verifier supervises the 27B policy with nearly constant soundness across verifier scales. Our contributions are:

- a formulation of agent self-improvement in which the verifier is a co-evolving, versioned component that filters self-generated data, attributes failures, and gates updates, so that verifier drift is measured rather than assumed away;
- GymHarness, an RSI framework whose recursive verifier is trained alongside the policy and retained only through an evidence-anchored ratchet over a hidden probe suite, with three-way failure attribution, minimal-privilege hints, and guarded checkpoint selection; and
- an evaluation on three in-domain and two held-out environments in which a 27B policy trained only by supervised fine-tuning improves across every benchmark, with ablations that isolate the recursive gate and analyses of rollout-audit rate, verifier scale, and drift.

2 Related Work

Self-training and data-centric self-improvement. STaR fine-tunes on self-generated rationales that reach correct answers and retries failed problems with the correct answer as a hint (Zelikman et al., 2022); ReST alternates growing a dataset from the current policy with offline improvement

(Gulcehre et al., 2023); and ReST-EM filters samples with binary feedback and surpasses fine-tuning on human data (Singh et al., 2023). For agents, CurateEvo rewrites executable data-curation code from failed trajectories on a held-out development set (Wang et al., 2026b), and benchmarks of data-centric self-improvement find that agents discover better data strategies but struggle to keep their gains (Meng et al., 2026; Wu et al., 2026b). These methods assume that the correctness filter or development signal is itself correct. GymHarness makes the filter a learned, co-evolving model and tests what keeps it correct.

Harness-level recursive self-improvement. SoL-Pi tests candidate harness changes against fixed capability tolerances and isolates held-out evaluation (Liu et al., 2026); ModularRSI evolves harness modules separately before integrating them (Wu et al., 2026a); SIFT searches over executable self-improvement proposals (Fu et al., 2026); RRSI regularizes proposal and selection so that evolved harnesses transfer beyond the evolve set (Xia et al., 2026); and Dream-RSI improves exploration strategies in a replay simulator built from discovery history (Zheng et al., 2026). PILOT, Recuris, and Argus evolve skills, memory, and persistent runtime state around fixed models (Li et al., 2026a; Xiao et al., 2026; Yu et al., 2026), while Harness-R1 trains a dedicated harness engineer from rerun outcomes (Shao et al., 2026). In these systems the signal that accepts an update is typically a fixed evaluator, such as capability tolerances, validation gates, benchmark scores, or rerun rewards. GymHarness keeps the runtime fixed and instead co-evolves the policy and its verifier; harness editing is complementary, and we include it as a reference.

Co-evolving and hardened verifiers. The Verification Horizon argues that no fixed reward remains effective as the policy grows and that verification must co-evolve with the generator (Wang et al., 2026a). EvoRubrics co-evolves a rubric generator with the policy under reinforcement learning (Ding et al., 2026); the Red Queen Gödel Machine co-evolves learned evaluators with self-improving agents and freezes the evaluation criterion within epochs (Iacob et al., 2026); DeepSeekMath-V2 labels new hard proofs with scaled verification to keep retraining its verifier, and checks the verifier with a learned meta-verifier (Shao et al., 2025); and Hacker–Fixer hardens benchmark verifiers with adversarial agents while solver checks protect legitimate solutions (Zhong et al., 2026). GymHarness adopts epoch freezing and adversarial probing, but accepts verifier updates through an execution-labeled probe suite rather than a learned or self-assessed judgment, which is what allows its recursion to terminate.

Environment and task evolution. EnvHarness reshapes static environments while retaining their original verifiers (Huang et al., 2026); RST, EvoEnv, and Environment Evolution synthesize or evolve executable tasks behind sandbox, staged, oracle, and invalid-test validation (Fan et al., 2026; Li et al., 2026b; Shi et al., 2026); and ARE scales environment and evaluation infrastructure for agents (Froger et al., 2025). These methods change what an agent practices on; GymHarness changes who evaluates it, and it reuses their oracle and no-op checks as anchors. Appendix F compares the frameworks along the objects they update and how their acceptance signals are validated.

3 GymHarness

GymHarness is an RSI framework for agents whose runtime is fixed and whose policy and verifier improve each other. We first define the setting (§3.1) and the loop that couples the two models (§3.2). We then describe the recursive verifier at the center of the framework (§3.3) and how policy updates are protected (§3.4). Appendix A gives the full algorithm, interfaces, and prompt contracts.

3.1 Setting

Two models around a fixed runtime. A policy acts in an environment through a fixed runtime that assembles its context, executes its actions, and returns observations. Following the ACE view of agentic data (Zeng et al., 2026), each episode yields an experience (E, q, τ, v): an environment, a task, a trajectory, and a verdict. GymHarness maintains two models. The *policy* π_k produces trajectories. The *verifier* V_k, a smaller generative model, reads a trajectory together with its ledger and returns a verdict with a rationale that cites the evidence it relies on. The index k counts self-improvement rounds, and both models are updated only by supervised fine-tuning (SFT).

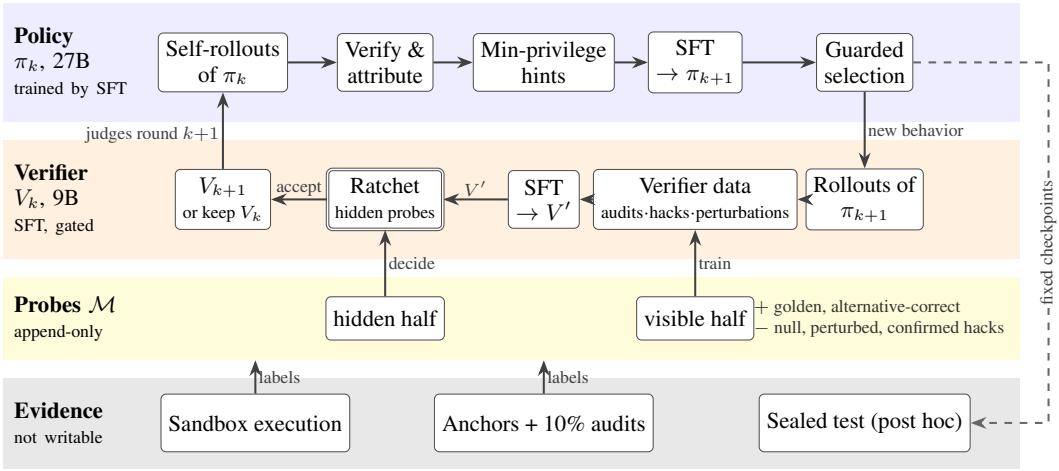

Figure 2: GymHarness. The policy loop (top) fine-tunes π_k on self-generated data accepted by V_k; the verifier loop (middle) retrains the verifier on the new policy's behavior. Each level is verified by the one below it: the policy by the verifier, the verifier by a hidden, append-only probe suite, and the probe suite by execution and a budget of ground-truth queries that the agent cannot write. The recursion thus ends in exogenous evidence. Quarantine of defective tasks and the research agent that attacks V_k are omitted for clarity.

Evidence is budgeted and outside the write surface. Each environment provides an executable success check, which we treat as ground truth. The loop reaches it only through fixed channels: *anchor* tasks with golden trajectories, *audits* of a small fraction of the trajectories rolled out in each round, a validation set for checkpoint selection, and sandboxed execution of verifier probes. A sealed test set is inaccessible until all model-selection decisions are final; afterward, checkpoints are evaluated on it offline for reporting only. Everything else the loop learns about success comes from V_k. The agent may propose new policy data, verifier data, and hints; it cannot change the runtime, the success checks, the anchors, the labels of the probe suite, or the sealed test (Table 4).

3.2 The self-improvement loop

Each round follows an observe–diagnose–revise–validate cycle over both models (Figure 2).

Observe. The runtime rolls out π_k on sampled training tasks and writes a ledger for every episode. The ledger separates observed facts, actions, the agent's stated hypotheses, and verdicts, so that later stages can trace a failure to an observation rather than to a scalar score.

Diagnose. V_k labels each trajectory, and an attributor routes every failure to one of three queues. A failure is a *task or environment defect* if re-execution changes its outcome, the golden trajectory fails, or the specification and the success check disagree; such tasks are quarantined. It is a *verifier defect* if V_k disagrees with an audit or anchor label; such cases become verifier training data. Only the remaining *capability gaps* drive new policy data. Without evidence, the attributor abstains.

Revise. Policy data consist of the self-generated trajectories that V_k accepts, together with hinted trajectories for near-zero-pass tasks (§3.4). Verifier data are built from the *next* policy's behavior: audited trajectories of π_{k+1}, attacks on V_k that execution confirms as failures, task-breaking perturbations of golden trajectories, and alternative correct solutions.

Validate. π_{k+1} replaces π_k only if a task-paired test on the validation set supports it; otherwise the loop rolls back. V_{k+1} replaces V_k only if it passes the recursive ratchet of §3.3; otherwise V_k is kept. Both models are then frozen for the next round, so each round's evaluation criterion stays fixed, as in epoch-based co-evolution of evaluators (Iacob et al., 2026).

3.3 THE RECURSIVE VERIFIER

Why the verifier must change, but not freely. A verifier is calibrated on the behavior of the policy it was trained against. After an SFT round the policy makes new kinds of mistakes, and the errors of a frozen verifier concentrate on exactly those behaviors, which are the ones that pass into the next round's training data. Because most useful data come from tasks the policy rarely solves, even a small false-positive rate can dominate the accepted data (Appendix D). Yet if the verifier were retrained on its own labels, or accepted whenever it looked better on its own training data, nothing would expose the errors it cannot see. GymHarness therefore lets the verifier learn from every new policy but separates who proposes a verifier update from what validates it.

The probe suite. Every probe is labeled by execution or by an anchor, never by a model. Positive probes are golden trajectories of anchor tasks and *alternative-correct* trajectories, which differ from the golden solution but pass the success check; they test whether the verifier accepts legitimate work expressed in new ways. Negative probes are null trajectories (empty or no-op), task-breaking perturbations of golden trajectories (a required step removed, a wrong object, a stale value), attacks on the current verifier that execution confirms as failures, and audited failures of the current policy. The suite is append-only and assigned by complete task-cluster $\times$ attack-family $\times$ generator-seed groups. Twenty percent of groups are shadow probes that never enter ratchet decisions; the remaining probes are split into a visible half for verifier training and a hidden half used only for acceptance, so that a verifier cannot pass by memorizing its own tests.

The ratchet. A candidate V' replaces the incumbent only if, on the hidden probes,

- *soundness does not regress*: its rejection rate on every negative class is at least the incumbent's, and no previously rejected probe is accepted;
- *completeness is non-inferior*: its acceptance of golden and alternative-correct probes is within one percentage point of the incumbent's; and
- *the change is real*: a paired bootstrap supports an improvement in at least one probe class.

Otherwise the candidate is discarded and the incumbent stays in place.

Why the recursion terminates. The verifier is improved by the same loop it supervises, and each of its improvements is certified one level down: the policy is judged by the verifier, the verifier by the probe suite, and the probe suite by execution. Unlike a learned meta-verifier (Shao et al., 2025), the last level contains no learned judgment that could itself drift, so there is nothing further to verify. This is the sense in which GymHarness's verifier is recursively verified.

Race margin. The ratchet protects against regressions on known probes; it does not say where the next blind spot is. To find it, we cluster the current policy's audited trajectories into behavior modes, split them into correct modes $\mathcal{C}_k$ and incorrect modes $\mathcal{H}_k$, and estimate the acceptance $a_k(m)$ of the verifier on each mode m. The *race margin*

$$\Delta_k = \max_{c \in \mathcal{C}_k} a_k(c) - \max_{h \in \mathcal{H}_k} a_k(h) \tag{1}$$

is positive when the verifier's most readily accepted behavior is correct, and negative when some reachable error is accepted more readily than every correct behavior, so that training would favor the error. GymHarness uses Δ_k to decide which modes the research agent should attack and audit next. It is a diagnostic, not an acceptance criterion.

3.4 GUARDED POLICY UPDATES

Verified self-generated data. The policy is fine-tuned on its own trajectories that the frozen verifier accepts, accumulated across rounds. No trajectory from a stronger model enters the policy's training data; golden trajectories reach it only through hints.

Minimal-privilege hints. On tasks the policy almost never solves, more samples rarely help and whatever does pass is suspect. GymHarness instead climbs a hint ladder: H1 states what the success

check requires (the end state, not how to reach it); H2 gives a structural sketch abstracted from a golden trajectory; H3 reveals a partial golden trajectory. It stops at the lowest level at which a sampled trajectory passes the verifier (and the success check, on anchor tasks). The accepted trajectory is then *de-privileged*: the hint is removed from its context before SFT, and trajectories that rely on information the task does not provide are discarded. This generalizes STaR's rationalization, which reveals the answer to failed problems (Zelikman et al., 2022). Hints are never issued for quarantined tasks, where they would teach the policy to satisfy a broken check.

Guarded selection. A new policy checkpoint is kept only if a task-paired bootstrap on the validation set shows an improvement at the 95% level; otherwise the loop rolls back to the previous checkpoint and records the rejected data. No sealed-test result enters these decisions.

4 EXPERIMENTS

We test whether a policy trained on its own verifier-filtered rollouts improves across benchmarks (§4.2), whether the recursive verifier is what makes this possible (§4.3), how many rollout audits and how large a verifier it needs (§4.4), and which components matter (§4.5).

4.1 SETUP

Environments. We train and evaluate on three interactive environments: WebShop, a simulated e-commerce site in which the agent searches for, customizes, and purchases products (Yao et al., 2022); ALFWorld, text-based household tasks aligned with ALFRED (Shridhar et al., 2020); and DBBench, database tasks solved through SQL, from AgentBench (Liu et al., 2023). Sealed test sets follow Shao et al. (2026): 500 WebShop tasks (goal seed 233), their released 500-task ALFWorld split, and the 300-task DBBench test set. Training tasks come from the remaining pools. To test transfer, we also evaluate on two environments that are never used for training or selection: ScienceWorld, an interactive environment for elementary-science experiments, on up to the first ten test variations of each of its 30 tasks (Wang et al., 2022); and the 144-task operating-system test set of AgentBench, which uses bash interaction in a sandbox (Liu et al., 2023). Table 7 lists the frozen versions and splits.

Models and budget. The policy is Qwen3.6-27B, fine-tuned with LoRA; the verifier is Qwen3.5-9B; a fixed Qwen3.5-397B serves as the research agent for attributions, attacks, and hints. The runtime is the AgentBench-based runtime released with Shao et al. (2026) and is never modified. The primary comparison runs for four rounds with 100 fixed anchor tasks per benchmark and an adaptive rollout audit on at most 10% of rolled-out tasks per round. The sealed test is inaccessible until all model-selection decisions are finalized; afterward, the final and intermediate checkpoints are evaluated offline for reporting only. Table 5 gives the settings; Appendix G gives the provenance, evidence, long-horizon, and uncertainty protocols.

Baselines. All methods start from the same policy and share the runtime and SFT recipe; rollout-based methods also share four primary rounds, the rollout budget, and, except for the oracle, the evidence budget. *Golden-SFT* trains once on anchor trajectories and environment demonstrations; *teacher-distilled SFT* trains on ground-truth-filtered trajectories of the 397B model, outside the evidence budget. Three baselines train on the policy's own trajectories: *self-judged SFT* lets the policy label its own data, *fixed-verifier SFT* freezes the initial verifier V_0, and *co-evolution without ratchet* retrains the verifier every round but accepts it without the hidden probe suite. The supplementary audit-only and accuracy/AUROC-gated controls use the same budget and are reported in Appendix G; *Oracle-filtered SFT* labels every rollout with ground truth and is an upper bound. As references we give the untrained policy and its Harness-R1 patch as published by Shao et al. (2026) for the same model, runtime, and test sets (held-out entries are ours).

Metrics. The primary metric is task success on the sealed test sets. We also report the precision of the self-generated data accepted in the final round, measured post hoc against ground truth, and, for verifiers, acceptance and rejection rates on hidden probes (Table 2). Appendix G separates ratchet-hidden probes from never-used shadow and fresh probes, reports the full evidence ledger, and gives paired bootstrap uncertainty intervals.

Table 1: Sealed success rate (%) of Qwen3.6-27B. In-domain environments supply training tasks; held-out environments are never used for training or selection. Prec. is the precision of the self-generated data accepted in the final round (–: no self-generated data; †: 1 by construction). Reference rows are reported by Shao et al. (2026) for the same model, runtime, and test sets; ∗: our evaluation. Bold and underline mark the best and second-best trained methods, excluding the oracle.

	In-domain (trained)				Held-out (never trained)			
Method	WebShop	ALFWorld	DBBench	Avg.	SciWorld	OS	Avg.	Prec.
Reference (no training)								
Untrained policy	43.4	70.6	69.7	61.2	41.2*	36.8*	39.0*	–
+ Harness-R1 patch	44.2	78.6	72.7	65.2	–	–	–	–
SFT on external supervision								
Golden-SFT	47.0	79.8	70.3	65.7	42.0	37.1	39.5	–
Teacher-distilled SFT	<u>52.0</u>	<u>86.8</u>	<u>74.0</u>	<u>70.9</u>	<u>45.2</u>	<u>39.8</u>	<u>42.5</u>	–
Self-improvement on policy-generated data (10% adaptive rollout-audit + fixed anchors)								
Self-judged SFT	44.0	72.2	68.9	61.7	40.6	36.2	38.4	0.61
Fixed-verifier SFT	48.6	81.4	71.0	67.0	43.1	37.9	40.5	0.71
Co-evolution w/o ratchet	46.8	78.2	70.1	65.0	41.8	36.0	38.9	0.74
GymHarness (ours)	**53.8**	**88.6**	**75.2**	**72.5**	**46.9**	**41.5**	**44.2**	0.95
Upper bound (ground truth on every rollout)								
Oracle-filtered SFT	55.0	90.2	76.4	73.9	47.6	42.3	45.0	1.00†

Table 2: Verifier quality after the final round, measured on hidden probes. Completeness columns are acceptance rates on golden and alternative-correct trajectories; soundness columns are rejection rates on null, perturbed, and hacked trajectories. Held-out hacks come from a stronger attacker and never enter the probe suite. Regress. is the share of validation tasks solved before a policy update and failed after it.

	Completeness		Soundness				Data
Verifier	Golden↑	Alt.-correct↑	Null↑	Perturbed↑	Held-out hack↑	Regress.↓	prec.↑
Policy as its own judge	0.91	0.80	0.83	0.66	0.35	11.2%	0.61
Frozen verifier V_0	0.98	0.83	0.99	0.84	0.58	6.8%	0.71
Co-evolved w/o ratchet	0.95	0.76	0.99	0.90	0.69	9.5%	0.74
GymHarness V_4	**0.99**	**0.92**	**1.00**	**0.96**	**0.93**	**2.1%**	0.95

4.2 Main results

Recursive verification improves every benchmark. Table 1 shows that GymHarness improves the policy on all five environments, by 10.4, 18.0, and 5.5 points on WebShop, ALFWorld, and DBBench, and by 5.2 points on average on the two held-out environments. It remains 1.4 points below oracle-filtered training, which labels every rollout, and recovers 89% of its improvement.

The bottleneck is verification, not supervision. Golden-SFT trains on scarce external demonstrations and reaches 65.7%. Distillation from a much larger teacher, filtered by ground truth, reaches 70.9%. GymHarness exceeds both with self-generated trajectories (hinted on near-zero-pass tasks; Table 22), while no baseline that uses a fixed or self-assessed verifier exceeds 67.0%.

4.3 The recursive gate prevents completeness loss during co-evolution

A frozen verifier degrades as the policy improves. Figure 1(b) tracks the precision of each verifier on the data it accepts. The frozen verifier starts at 0.86 and falls to 0.71 as the policy trains, and the policy gains flatten after the second round (Figure 1(a)). The verifier itself stayed unchanged; the policy moved into behaviors it was never calibrated on, and Table 2 shows that it rejects only 0.58 of held-out hacks. A policy that judges itself is worse still: the precision of its accepted data never exceeds 0.70.

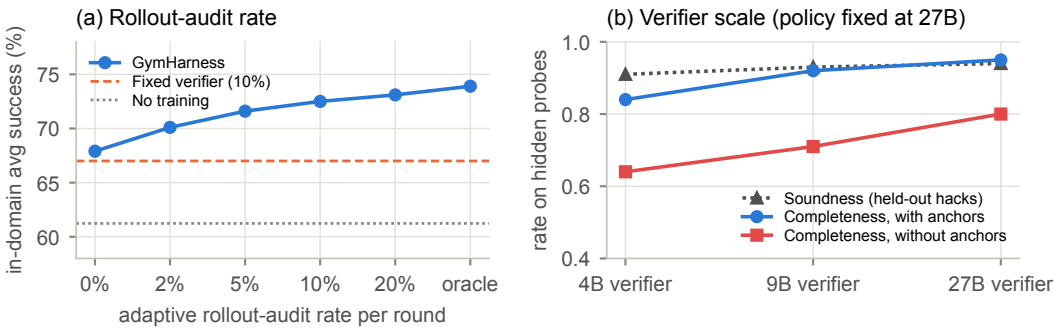

Figure 3: (a) In-domain success as the adaptive rollout-audit rate grows, with anchor, validation, and probe checks held fixed; "oracle" labels every rollout. (b) Verifier quality across Qwen3.5-4B, Qwen3.5-9B, and Qwen3.6-27B, with a fixed Qwen3.6-27B policy. Soundness is nearly flat, whereas completeness depends on capability unless anchors are available.

The ratchet preserves completeness while hardening the verifier. Retraining the verifier every round without the hidden probe suite accepted all 12 candidate verifiers. This ungated variant rejects the trained negative classes well (0.90 on perturbations and 0.69 on held-out hacks), but its alternative-correct acceptance falls to 0.76 and its post-update regression increases to 9.5%, so fewer and less diverse trajectories reach policy SFT. It ends 2.0 points below the frozen verifier despite its higher final data precision. GymHarness accepted 7 of 12 candidates, raised held-out hack rejection to 0.93 while keeping alternative-correct acceptance at 0.92, and lowered post-update regressions to 2.1%. The audit appendix reports per-round accepted counts and difficulty strata. Figure 4 in the appendix shows the same pattern within ALFWorld alone.

4.4 A SMALL VERIFIER AND A SMALL ROLLOUT-AUDIT BUDGET SUFFICE

Rollout-audit budget. Figure 3(a) varies only the adaptive rollout-audit rate; the other evidence channels stay fixed, and Appendix G.2 crosses audit rates with and without anchors. Without adaptive audits (0%), GymHarness reaches 67.9%, already above fixed-verifier training with a 10% audit rate (67.0%). The default 10% audit reaches 72.5%, and returns diminish beyond it (73.1% at 20%, 73.9% with the oracle).

A weaker verifier can supervise a stronger policy. Figure 3(b) varies the verifier from Qwen3.5-4B through Qwen3.5-9B to Qwen3.6-27B while keeping the Qwen3.6-27B policy fixed. Held-out hack rejection changes little (0.91 to 0.94), consistent with weak supervisors generalizing beyond their own capability (Burns et al., 2023) and with weak hacker–fixer loops defending against stronger attackers (Zhong et al., 2026). Completeness behaves differently. Without anchors, acceptance of alternative-correct solutions rises from 0.64 to 0.80 with verifier size: recognizing a legitimate solution requires capability. Anchors close most of this gap (0.84 for the 4B verifier), which is why the probe suite contains golden and alternative-correct trajectories rather than only attacks.

4.5 ABLATIONS

Table 3 removes one component at a time (Table 9 in the appendix lists all twelve ablations). The recursive ratchet matters most (7.5 points), followed by co-evolution itself (5.5), anchor calibration (3.7), and the hidden probe split (3.6); a verifier that sees its own acceptance tests learns them. Among the policy-side components, de-privileging matters most on held-out environments (41.0% without it, against 44.2%): a policy trained with the hint in its context learns to depend on information that new tasks do not provide.

5 ANALYSIS

The race margin warns before the verifier drifts. Mean verifier accuracy can hide a single error that is accepted more readily than any correct behavior. In a controlled simulation with finite be-

Table 3: Ablations of GymHarness (sealed success, %; in-domain drop in parentheses). Each row removes one component and keeps everything else fixed.

Variant	In-domain	Held-out	Data prec.
GymHarness (full)	72.5	44.2	0.95
− recursive ratchet (visible acceptance)	65.0 (−7.5)	38.9	0.74
− verifier co-evolution (frozen V_0)	67.0 (−5.5)	40.5	0.71
− hidden probe split	68.9 (−3.6)	41.3	0.83
− adversarial hack probes	69.4 (−3.1)	41.6	0.84
− anchor calibration	68.8 (−3.7)	41.8	0.88
− three-way failure attribution	69.9 (−2.6)	42.4	0.90
− de-privileging (hint kept in context)	69.8 (−2.7)	41.0	0.94

havior modes (Appendix E), the sign of the race margin (Eq. 1) predicts which runs end dominated by an incorrect mode with an AUROC of 0.91, against 0.63 for mean acceptance and 0.68 for the aggregate false-positive rate, and it turns negative 3.1 rounds before the error mode overtakes. In the real fixed-verifier and ungated co-evolution runs (three seeds each), the margin became negative before the precision drop in 5 of 6 runs. GymHarness thus uses the margin to schedule attacks and audits on the modes that are winning, rather than as a gate.

Full ledgers make failures attributable. On 200 failures audited by two reviewers with independent replay, the three-way attribution (task defect, verifier defect, capability gap) is correct for 81% of failures when the attributor reads the full fact/action/hypothesis/verdict ledger, against 46% when it sees only the final outcome and score; it abstains on 9%. Misattribution is costly in both directions: a verifier defect treated as a capability gap produces policy data that teach the error, and a broken task treated as a capability gap invites hints that teach the policy to satisfy the broken check.

Minimal hints unlock near-zero tasks without leaking. On the 18% of training tasks with near-zero initial pass rate (after quarantining 6.4% of them as defects), 32 unhinted samples yield a verified pass on only 7% of tasks (Table 8 in the appendix). The minimal-privilege ladder raises coverage to 42% with 3% leakage. Always revealing a partial golden trajectory covers more tasks (49%) but leaks into 17% of accepted trajectories and helps the stratum less (+9.6 vs. +14.8 points).

The gains do not originate from the research agent. A 397B research agent writes attributions, attacks, and hint abstractions, but never contributes trajectories to policy SFT; its verifier-quality and hint-provenance audit is in Appendix G.8. When the 27B policy plays these roles itself, GymHarness still reaches 70.4% in-domain and 43.1% held-out, against 66.1% for fixed-verifier training in the same setting. Most of the improvement thus comes from the recursive verifier rather than from a stronger model in the loop. The recursion costs 21% more compute than fixed-verifier training (13% for verifier fine-tuning and 8% for probes and replay), including rejected candidates and rollbacks.

6 Conclusion and Limitations

GymHarness co-evolves the verifier that labels self-generated data with the policy, while a hidden, append-only probe suite labeled by execution certifies every verifier update. A 27B agent trained on its own verifier-filtered rollouts, with golden-derived hints only on near-zero-pass tasks, improves across every in-domain and held-out environment; a frozen verifier loses precision and ungated co-evolution loses completeness and downstream quality.

Claims remain bounded by finite probe families and executable evidence. Concretely, an error mode that no probe family represents can pass the ratchet until an audit or attack adds it to the suite, and environments without an executable success check, such as open-ended writing, lie outside the current design. The four-round comparison and the eight-round ALFWorld audit in the appendix do not demonstrate open-ended recursive self-improvement.

AI USE STATEMENT

Generative AI tools were used to organize the literature, to draft and edit prose, and to assist with code for the experiment infrastructure. Every citation was checked against the full text of the cited paper, every claim about prior work is traced to a specific section of that paper, and all code used for the reported experiments was reviewed by the authors. Language models also appear inside the method itself, as the trained policy, the trained verifier, and the fixed research agent; their roles are specified in Section 4.1 and Appendix A, and no model output is used as ground truth. The authors take full responsibility for the content of this paper.

ETHICS STATEMENT

This work does not involve human subjects or personal data. All agent actions run in sandboxed environments, and the evaluated tasks do not reach external systems. The main risk studied here is evaluator gaming: a self-improving system can learn to satisfy its verifier instead of the task. GymHarness keeps the runtime, the success checks, the probe labels, and the sealed test outside the agent's write surface, and it reports soundness, completeness, and regressions together with task success. The attack trajectories collected in the probe suite could help others exploit the same environments, so we will release them under a research-use agreement.

REPRODUCIBILITY STATEMENT

Section 4.1 and Appendices A–B specify the environments, models, evidence budget, and acceptance rules; Table 5 lists every training and inference setting, and Table 7 pins each environment and the runtime to a repository commit together with its evaluation split. Every number in the paper is generated by a script from a single results file that records the source and status of each value. After the review period we will release this file, the configuration file, the generation scripts, the task splits, the probe suite and its generators, the verifier and attributor prompts, and the per-seed, per-task verdicts, including rejected verifier candidates and rolled-back policy checkpoints.

AUTHOR CONTRIBUTIONS

All authors contributed to the study design, implementation, analysis, and manuscript preparation.

ACKNOWLEDGMENTS

We thank the open-source benchmark and agent communities for releasing the environments used in this study.

REFERENCES

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeff Wu. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision, 2023. URL https://doi.org/10.48550/arXiv.2312.09390.

Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Zheng Li, Jinyang Zhang, Zhijing Wu, Junfeng Zhao, and Yasha Wang. EvoRubrics: Dynamic rubrics as rewards via adversarial co-evolution for LLM reinforcement learning, 2026. URL https://doi.org/10.48550/arXiv.2606.23038.

Zhiyuan Fan, Tinghao Yu, Yuanjun Cai, Jiang Zhou, Jiangtao Guan, Jincheng Liu, Yun Yang, Dingxin Hu, Zhuo Han, Xing Wu, Feng Zhang, and Lilin Wang. Environment evolution for terminal agents, 2026. URL https://doi.org/10.48550/arXiv.2609.04128.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine

Benhalloum, Grégoire Mialon, and Thomas Scialom. ARE: Scaling up agent environments and evaluations, 2025. URL https://doi.org/10.48550/arXiv.2509.17158.

Xinghong Fu, Aravinth Kulanthaivelu, and Yutaro Yamada. Self improvement via fast tree-search, 2026. URL https://doi.org/10.48550/arXiv.2609.19526.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling, 2023. URL https://doi.org/10.48550/arXiv.2308.08998.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi Mishra, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. EnvHarness: Awakening static worlds for agent learning, 2026. URL https://doi.org/10.48550/arXiv.2608.19880.

Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Ambroise Odonnat, Zeyu Cao, Bill Marino, Xinchi Qiu, and Nicholas D. Lane. The red queen Gödel machine: Co-evolving agents and their evaluators, 2026. URL https://doi.org/10.48550/arXiv.2606.26294.

Boxiu Li, Zimo Wen, Yijia Fan, Chuan Wen, Fan Yang, Hangxi Guo, Jiaao Wu, Jiachen Zhang, Junxiang Lei, Mukai Li, Ruize Tang, Runjing Gu, Shibo Hu, Sihan Chen, Sufeng Guo, Wanbo Zhang, Xian Zhang, Xiaoyu Chen, Xuanhe Zhou, Xuyao Huang, Yifei Gao, Yifei Shen, Yilin Chen, Yuheng Wu, Yuzhe Zhang, Zelong Zhao, and Zhijie Deng. Argus: A general-purpose agentic reasoning runtime for long-horizon tasks, 2026a. URL https://doi.org/10.48550/arXiv.2608.05144.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks, 2026b. URL https://doi.org/10.48550/arXiv.2608.05466.

Haozhe Liu, Tian Ye, Sensen Gao, Qihang Cao, Yitong Li, Mingchen Zhuge, Duomin Wang, Ruihua Zhang, Ping Luo, Jiawang Bian, Lei Zhu, Ligeng Zhu, Enze Xie, and Song Han. SoL-Pi: Recursively scaling auto-research loops for efficient agent harness, 2026. URL https://doi.org/10.48550/arXiv.2609.20519.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents, 2023. URL https://doi.org/10.48550/arXiv.2308.03688.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. RSIBench-Data: Benchmarking data-centric research for recursive self-improvement, 2026. URL https://doi.org/10.48550/arXiv.2607.25886.

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, 2026. URL https://doi.org/10.48550/arXiv.2608.02276.

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning, 2025. URL https://doi.org/10.48550/arXiv.2511.22570.

Yucheng Shi, Zhenwen Liang, Kishan Panaganti, Dian Yu, Wenhao Yu, and Haitao Mi. Learning to build the environment: Self-evolving reasoning RL via verifiable environment synthesis, 2026. URL https://doi.org/10.48550/arXiv.2605.14392.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning, 2020. URL https://doi.org/10.48550/arXiv.2010.03768.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models, 2023. URL https://doi.org/10.48550/arXiv.2312.06585.

Binghai Wang, Chenlong Zhang, Dayiheng Liu, Jiajun Zhang, Jiawei Chen, Mingze Li, Mouxiang Chen, Rongyao Fang, Siyuan Zhang, Xuwu Wang, Yuheng Jing, Zeyao Ma, and Zeyu Cui. The verification horizon: No silver bullet for coding agent rewards, 2026a. URL https://doi.org/10.48550/arXiv.2606.26300.

Dingzirui Wang, Xuanliang Zhang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. CurateEvo: Data-curation evolving for agentic post-training, 2026b. URL https://doi.org/10.48550/arXiv.2607.06140.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader?, 2022. URL https://doi.org/10.48550/arXiv.2203.07540.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. ModularRSI: Modular and generalizable recursive harness self-improvement, 2026a. URL https://doi.org/10.48550/arXiv.2609.14857.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Yuxuan Zhang, Xinping Lei, Junting Zhou, Zexuan Wang, Yuchen Wu, Huan Zhou, Duo Wang, Yinzhu Piao, Yongchang Peng, Yunfeng Shi, Jin Chen, Zuo Wang, Jinkai Liu, Jiaheng Liu, Wenxuan Zhang, Shen Yan, Wenhao Huang, and Ge Zhang. Aspire: Can models self-evolve from vague goals?, 2026b. URL https://doi.org/10.48550/arXiv.2608.31111.

Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. RRSI: Regularized recursive self-improvement of agent harnesses, 2026. URL https://doi.org/10.48550/arXiv.2609.24972.

Yang Xiao, Yusong Sun, Haoyi Wu, Wenyang Hui, Wen Da, Zhaokai Luo, Mu Chuan, Yao Hu, Wenjie Li, and Chengyue Jiang. PILOT in the loop: Live self-improvement for long-horizon agents, 2026. URL https://doi.org/10.48550/arXiv.2608.26530.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents, 2022. URL https://doi.org/10.48550/arXiv.2207.01206.

Zhaochen Yu, Yingcheng Wu, Zhenfei Yin, Kaiyuan Chen, Zhe Zhao, Mengdi Wang, Shuicheng Yan, and Ling Yang. Recursive experiential-working memory evolution for long-horizon agent harnesses, 2026. URL https://doi.org/10.48550/arXiv.2608.24876.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning, 2022. URL https://doi.org/10.48550/arXiv.2203.14465.

Xingshan Zeng, Zishan Xu, Boju Zhang, Yuzhou Wu, Lingzhi Wang, Jianghao Lin, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Weinan Zhang, Yong Yu, Qun Liu, and Weiwen Liu. What makes good agentic data? an ACE lens on data generation for LLM agents, 2026. URL https://doi.org/10.48550/arXiv.2608.27260.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, Xue Wang, Yue Zhuan, Wang-Cheng Kang, Renkai Xiang, Heng Huang, Xinwu Cheng, and Yunsong Guo. Dream-RSI: Recursive self-improvement through evolving worlds, 2026. URL https://doi.org/10.48550/arXiv.2609.14858.

Ziqian Zhong, Ivgeni Segal, Ivan Bercovich, Shashwat Saxena, Kexun Zhang, and Aditi Raghunathan. Hardening agent benchmarks with adversarial hacker-fixer loops, 2026. URL https://doi.org/10.48550/arXiv.2606.08960.

A IMPLEMENTATION

This appendix describes the GymHarness loop, its interfaces, and the write surface of the agent. The runtime that executes the policy is fixed throughout; only the policy and the verifier are trained, and both only by supervised fine-tuning.

A.1 ALGORITHM

Algorithm 1 GymHarness: co-evolution of a policy and a recursively verified verifier.

Require: policy π_0; fixed runtime; evidence budget (anchors, audit fraction, validation set); rounds K

build probes from anchors by execution (golden, perturbed, null); assign complete task-cluster $\times$ attack-family $\times$ generator-seed groups; reserve 20% as shadow and split the remainder into visible and hidden halves of $\mathcal{M}$
$V_0 \leftarrow$ SFT of the verifier on the visible probes
for $k = 0, \ldots, K-1$ **do**
    roll out π_k on training tasks and write a ledger for every episode
    V_k labels each trajectory; the attributor routes failures to task defect, verifier defect, capability gap, or abstain
    policy data $\leftarrow V_k$-accepted trajectories $\cup$ de-privileged hinted trajectories for near-zero-pass tasks
    $\pi' \leftarrow \mathrm{SFT}(\pi_k, \text{policy data})$; keep π' as π_{k+1} only if a paired validation test favors it, else roll back
    audit a budgeted fraction of π_{k+1}'s rollouts; the research agent attacks V_k; execution confirms failures
    append confirmed probes to $\mathcal{M}$ by the pre-registered group split; shadow groups never enter the ratchet
    $V' \leftarrow \mathrm{SFT}(V_k, \text{visible probes, audits, and verifier defects})$
    $V_{k+1} \leftarrow V'$ if the ratchet passes on the hidden probes, else $V_{k+1} \leftarrow V_k$
    estimate the race margin of V_{k+1} on π_{k+1}'s modes to choose the next attacks and audits
end for
after all selection decisions are final, evaluate $\pi_0, \ldots, \pi_K$ offline on the sealed test sets, for reporting only
return π_K

A.2 RUNTIME INTERFACE

An environment adapter implements the following interface for a task q. The output of `check`, not a model score or a log, is the sole source of ground truth. If an adapter cannot produce an executable outcome, the case is marked `abstain` and excluded from probes and verifier training.

Operation	Input	Output
`reset`	task id, seed, environment version	initial observation, run id
`step`	run id, action	observation, tool result, side-effect record
`finish`	run id	terminal state, trace hash
`check`	task id, trace or artifact	evidence label, check log, check version
`replay`	task id, trace hash, seed	repeated outcome, determinism flag

A.3 LEDGER, VERIFIER INTERFACE, AND PROMPT CONTRACTS

Each ledger row records the run id, task id, environment hash, seed, turn index, observation hash, action, tool result, record type, evidence ids, and verifier version. Four record types are required: `fact` (observation, source), `action` (action, precondition, result), `hypothesis` (cause, confidence, falsifier), and `verdict` (verifier, check id, route, evidence). The verifier reads a trajectory with its ledger and returns a binary verdict, a list of the evidence ids it relies on, and a short rationale; a verdict without evidence ids is treated as a rejection.

Every model call follows a typed contract. The *policy* receives the task, the tool schema, the budget, and, during data construction only, a hint; it never writes verdicts. The *attributor* receives a full ledger and the available checks and returns one route with evidence ids and a falsifying test; a

capability route without evidence is invalid. The *attacker* receives the current verifier's interface and a task and tries to produce a trajectory that the verifier accepts while the success check rejects it; only execution decides whether an attack succeeded. The *hint abstractor* turns a golden trajectory into an H1 or H2 hint and may not copy actions verbatim.

A.4 Write surface and defaults

Table 4: Write surface of the self-improvement loop. The agent proposes data and models; acceptance always relies on objects it cannot write.

Object	Read	Write	Accepted by
Policy weights	–	new SFT data	paired validation test
Verifier weights	–	new verifier data	ratchet on hidden probes
Hints, policy data	yes	propose	de-privileging, leakage filter
Ledger	own runs	append	evidence ids required
Probes, visible half	yes	append if confirmed	–
Probes, hidden half	no	append if confirmed	–
Checks, anchors, audits	within budget	no	–
Runtime, sealed tests	no	no	–

Unless stated otherwise, the primary comparison uses four rounds on the three in-domain environments. The supplementary long-horizon protocol uses eight ALFWorld rounds with seeds 0, 1, and 2; WebShop and ALFWorld uncertainty checks add seeds 3 and 4. Each benchmark has a fixed anchor set and validation set recorded in the manifest; adaptive rollout audits query at most 10% of the round's rollouts. Completeness gates use a one-percentage-point non-inferiority margin and task-paired bootstrap confidence intervals at the 95% level. Numerical results are emitted only after the corresponding run log has been validated. Policy and verifier are fine-tuned with the configuration in the run manifest; rollouts use the runtime's default decoding settings, and sealed evaluation uses temperature zero.

Table 5 lists the training and inference configuration. The policy and verifier are re-trained from their base models on data accumulated so far; exact values are read from each run manifest.

B Environments and verifier instantiation

Held-out environments are used only for final evaluation. Their versions and task lists are frozen in the experiment manifest, and no trajectory from them enters any policy data, verifier data, or probe.

Table 7 records the frozen source versions and evaluation splits. Held-out task lists are hashed in the manifest before training and are inaccessible to data construction.

C Additional results

D Why small false-positive rates matter

Let a generator solve a task cluster with probability p, and let the verifier accept correct trajectories with probability β (completeness) and incorrect ones with probability α (false-positive rate). By Bayes' rule, the fraction of accepted trajectories that are correct is

$$\mathrm{Prec}(p;\alpha,\beta)=\frac{\beta p}{\beta p+\alpha(1-p)},\qquad\text{and}\qquad \mathrm{Prec}\geq\rho\iff\alpha\leq\frac{\beta p(1-\rho)}{\rho(1-p)}. \tag{2}$$

At $p=0.05$ and $\beta=0.9$, a verifier with $\alpha=0.05$ accepts data that are about half wrong ($\mathrm{Prec}=0.49$), and 90% precision requires $\alpha\leq 0.53\%$. Tasks the policy rarely solves are the ones whose data matter most, which is why GymHarness both hardens the verifier and raises p with hints.

Table 5: Training, inference, and evaluation configuration.

	Policy	Verifier
Model	Qwen3.6-27B	Qwen3.5-9B
Fine-tuning	LoRA (r=32, α=64, dropout 0.05)	full-parameter SFT
Optimizer, schedule	AdamW, cosine, warmup 0.03	AdamW, cosine, warmup 0.03
Learning rate	5×10^{-5}	1×10^{-5}
Epochs per round	2	2
Global batch size	24	24
Maximum sequence length	24,576	32,768
Initialization each round	base model, all verified data	base model, all verifier data
Precision / thinking	BF16 / disabled	BF16 / –
Rollouts, inference, and evaluation		
Rounds	4 primary rounds; 8-round ALFWorld audit; curves are post-hoc checkpoint replays	
Training tasks per round	1,000 per in-domain environment, 4 samples each	
Rollout sampling	temperature 0.7, top-p 0.95, at most 4,096 tokens per turn	
Hint ladder	at most 8 samples per hint level	
Verifier decoding	temperature 0, at most 2,048 tokens	
Research agent	Qwen3.5-397B; temperature 0 (attribution, hints), 0.7 (attacks); 50 attacks per environment and round	
Evidence budget	100 anchors per environment; adaptive rollout audits on 10% of rollout tasks per round	
Probe provenance	groups: task cluster x attack family x generator seed; 20% shadow probes held out from ratchet; fresh-final=true	
Sealed evaluation	independent evaluator after all checkpoints are fixed	
Validation sets	100 / 99 / 100 tasks (WebShop / ALFWorld / DBBench)	
Statistical tests	task-paired bootstrap, 10,000 resamples, 95% level; non-inferiority margin 1 pp	
Evaluation decoding	temperature 0, at most 4,096 tokens per turn, thinking disabled	
Seeds and hardware	primary seeds 0, 1, 2; uncertainty seeds 3, 4; training seed 42; one node with 8 NVIDIA H800 GPUs per training stage	

Table 6: How each in-domain environment instantiates evidence and probes. Success checks are the environments' own; perturbation and attack classes are our design.

Environment	Ground truth	Perturbed negatives	Attack classes
WebShop	purchased item satisfies all instruction attributes and options	wrong option selected; item over budget; purchase before options are set	claim success after a partial match; stop at the first plausible item
ALFWorld	goal condition reached in the final state	required transformation skipped; object placed in the wrong receptacle	report completion without the final placement; act on a look-alike object
DBBench	returned answer or final database state matches the reference	mutation with a stale or reformatted value; query on the wrong table	answer copied from the task text; superset query presented as exact

E RACE-MARGIN SIMULATION

The simulation uses a finite set of correct and incorrect behavior modes and a selector that moves probability mass toward accepted samples after each round. We construct pairs of verifiers with the same mean acceptance on the initial mode mixture but different acceptance of their worst incorrect mode, vary the acceptance levels and initial mode frequencies across runs, and record whether an incorrect mode eventually dominates. The race margin of Eq. 1, mean acceptance, and aggregate false-positive rate are computed on the initial mixture and compared as predictors of domination (AUROC) and by lead time. The simulation tests a selection mechanism alone; it is not a model of SFT on long-horizon trajectories.

Table 7: Frozen environment versions and evaluation splits.

Environment	Source (version)	Evaluation split
Runtime	`DeepExperience/Harness-R1` @ `94f2e087f573`	Harness-R1 runtime (AgentBench-based controller)
WebShop	`princeton-nlp/WebShop` @ `64fa2a5c15c7`	task indices 0–499, goal seed 233 (500 tasks)
ALFWorld	`alfworld/alfworld` @ `aaba6870f86c`	Harness-R1 test_set.json: 109 new_std + 391 stratified train_valid, seed 20260609 (500 tasks)
DBBench	`THUDM/AgentBench` @ `d1e4a10db08c`	standard DB test set (300 tasks)
ScienceWorld	`allenai/ScienceWorld` @ `e8216d6044e8` (release 1.2.3)	30 tasks; up to the first 10 test variations of each task; success = task completed (held out)
AgentBench OS	`THUDM/AgentBench` @ `d1e4a10db08c`	standard OS-interaction test set (144 tasks); success rate (held out)

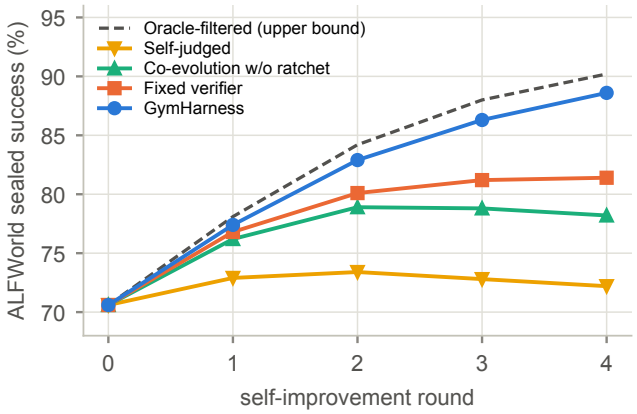

Figure 4: Sealed ALFWorld success over self-improvement rounds. The ordering of the methods matches the in-domain average in Figure 1.

Table 8: Hints on near-zero-pass training tasks. Coverage is the share of tasks with at least one verified pass; leakage is the share of accepted trajectories that use information unavailable without the hint; stratum gain is the sealed-success improvement on the corresponding test tasks.

Near-zero tasks	Coverage (%)	Leakage (%)	Gain
32 samples, no hint	7	0	+2.1
H1 checks	22	1	+9.8
H2 structure	34	3	+12.9
H3 partial golden	49	17	+9.6
Minimal privilege	42	3	+14.8

F Comparison with related frameworks

G Audit protocols and reproducibility

This section defines the provenance, evidence accounting, controls, and reproducibility records for supplementary audits. The runner writes an immutable manifest before the first rollout. The numerical tables below are generated from `results/audits.yaml`; external runs use the same aggregation interface.

Table 9: All ablations of GymHarness (sealed success, %; in-domain drop in parentheses).

Variant	In-domain	Held-out	Data prec.
GymHarness (full)	72.5	44.2	0.95
− recursive ratchet (visible acceptance)	65.0 (−7.5)	38.9	0.74
− verifier co-evolution (frozen V_0)	67.0 (−5.5)	40.5	0.71
− hidden probe split	68.9 (−3.6)	41.3	0.83
− adversarial hack probes	69.4 (−3.1)	41.6	0.84
− anchor calibration	68.8 (−3.7)	41.8	0.88
− alternative-correct probes	70.2 (−2.3)	42.9	0.93
− three-way failure attribution	69.9 (−2.6)	42.4	0.90
− hint ladder	70.6 (−1.9)	43.0	0.95
− de-privileging (hint kept in context)	69.8 (−2.7)	41.0	0.94
− guarded checkpoint selection	70.9 (−1.6)	43.1	0.95
− full ledger (scalar feedback)	71.2 (−1.3)	43.5	0.94
− epoch freeze (update every batch)	71.0 (−1.5)	43.3	0.92

Table 10: What each framework updates and how its evaluator is maintained, based on the cited papers.

Framework	Updated objects	How the evaluator changes
STaR, ReST-EM (Singh et al., 2023; Zelikman et al., 2022)	policy	fixed answer check
CurateEvo (Wang et al., 2026b)	data-curation code, policy	fixed development set
SoL-Pi, RRSI (Liu et al., 2026; Xia et al., 2026)	harness	fixed tolerances or verifier
Harness-R1 (Shao et al., 2026)	harness engineer, harness	fixed rerun reward
Hacker–Fixer (Zhong et al., 2026)	benchmark verifier	patched; solver checks protect valid solutions
EvoRubrics (Ding et al., 2026)	policy, rubric generator	co-trained by reinforcement learning
Red Queen Gödel Machine (Iacob et al., 2026)	agents, learned evaluators	co-evolved; fixed within epochs
DeepSeekMath-V2 (Shao et al., 2025)	prover, verifier	retrained; checked by a learned meta-verifier
GymHarness	policy, verifier	retrained; accepted by an execution-labeled probe suite

G.1 Sealed access and probe provenance

Each probe is keyed by task cluster, attack family, generator seed, source check version, and content hash. Complete groups are assigned before verifier updates: 20% form a shadow split that never enters ratchet decisions, while the remaining groups form the ratchet-hidden split. Fresh-final attacks are generated only after the verifier checkpoint is frozen by the independent 397B research agent and 27B self-researcher; fresh trajectories are never replayed into training prompts. A sealed evaluator is callable only by the independent final-evaluation process. Training sees aggregate pass/fail and a fixed failure code, never probe text or per-probe scores. Curves at checkpoints 0, 4, and 8 are post-hoc replays after all checkpoint decisions are fixed. The access log records process identity, checkpoint hash, evaluator phase, and timestamp.

Table 11: Immutable evaluation groups and permitted feedback.

Group	Construction	Consumer	Feedback
dev-validation	fixed tasks and checks	policy gate	paired aggregate
ratchet-hidden	execution-labeled probes outside shadow	verifier gate	aggregate and failure code
shadow	held-out cluster–attack groups	final evaluator	none during training
fresh-final	generated after verifier freeze	one-time final evaluator	none before finalization

Table 12: Probe provenance and independent-attack audit.

Probe group	Golden	Alt.	Null	Perturb.	Hack	ECE	Brier	N
Ratchet hidden	99.0	92.0	100.0	96.0	93.0	0.030	0.040	300
Shadow group	98.0	91.0	99.0	95.0	91.0	0.036	0.047	300
Fresh final	98.0	90.0	99.0	94.0	90.0	0.042	0.054	300

G.2 Evidence ledger and anchor–audit decomposition

The ledger counts every source of executable evidence; the 10% rate of the main text covers only the adaptive rollout audits. Each benchmark, seed, and round emits JSONL records; the aggregator derives rates only from explicit events. The ALFWorld matrix crosses 0, 5, 10, and 20% adaptive audits with 0 and 100 fixed anchors; the 100% audit endpoint is an oracle reference and never tunes the default. WebShop and DBBench repeat zero-audit, anchors-only, and default-10% endpoints. Reports include accepted ratio, difficulty-stratified accepted trajectories, post-update regression, candidate/accepted/rollback counts, probe-suite size, and execution/model-token cost.

Table 13: Evidence-ledger schema emitted per benchmark, seed, and round.

Block	Required fields	Normalization
Anchors	task, check, golden trace, outcome	fixed budget
Validation	candidate, task, outcome, gate decision	per candidate and round
Rollout audit	trajectory, query reason, outcome	per 100 rollouts
Probes	class, generated, executed, reused, outcome	per verifier update
Hints	stratum, tokens, trajectory, leakage decision	per accepted trajectory
Updates	candidate, accepted, rollback, checkpoint hash	per round
Costs	execution calls, model calls, tokens, wall time	per accepted trajectory

Table 14: Anchor and adaptive-audit endpoint matrix.

Environment	Anchors	Audit (%)	Labels/100	Success	Transfer avg.
ALFWorld	0	0	0	66.2	40.2
ALFWorld	0	5	5	69.5	42.1
ALFWorld	0	10	10	71.4	43.5
ALFWorld	0	20	20	72.1	43.9
ALFWorld	100	0	0	68.1	41.0
ALFWorld	100	5	5	70.4	42.4
ALFWorld	100	10	10	72.5	44.2
ALFWorld	100	20	20	73.2	44.7

Table 15: Evidence ledger accounting per 100 rollouts.

Ledger field	Amount	Denominator
Anchor checks	100	per 100 rollout episodes
Validation checks	100	per 100 rollout episodes
Adaptive audit checks	10	per 100 rollout episodes
Probe execution checks	64	per 100 rollout episodes
Hint trajectories	42	per 100 rollout episodes
Execution calls	274	per 100 rollout episodes
Model calls	324	per 100 rollout episodes
Policy tokens (M)	18.4	per 100 rollout episodes
Verifier tokens (M)	6.1	per 100 rollout episodes

G.3 Direct audit and selector controls

Audit-only SFT uses the same rollouts, anchors, and execution budget as GymHarness but retains only successes confirmed by adaptive audits. It never uses verifier-accepted uncertified trajectories. *Accuracy-gated co-evolution* selects candidate verifiers by hidden-probe accuracy and, separately, AUROC, without class-wise non-inferiority, previously-rejected probe, or bootstrap-ratchet constraints. Both controls share prompts, seeds, rollout counts, and checkpoint rules with GymHarness. The ledger records accepted trajectories per round and difficulty quantile, golden and alternative-correct completeness, negative-class rejection, and regression. ALFWorld also uses its standard

unseen task split; template and layout IDs are recorded for train, validation, test, and unseen tasks. Verifier-scale claims include downstream policy success for 4B, 9B, and 27B settings.

G.4 LONG-HORIZON STABILITY

The supplementary stability run uses eight ALFWorld rounds with seeds 0, 1, and 2. Checkpoints 0, 4, and 8 are replayed by the independent evaluator only after training and selection are complete. Each round records policy success, accepted-data precision, golden and alternative-correct completeness, known and held-out hack rejection, accepted ratio, regression, candidate/accepted/rollback counts, probe-suite size, and execution/model-token cost. Any stability statement is limited to this eight-round horizon.

Table 16: Eight-round ALFWorld stability.

Round	ALFWorld	Transfer	Prec.	Golden	Alt.	Known	Fresh	Regress.
0	61.2	39.0	86.0	91.0	80.0	58.0	44.0	6.8
4	72.5	44.2	95.0	99.0	92.0	93.0	87.0	2.1
8	74.1	45.3	96.0	99.0	93.0	94.0	90.0	1.8

Table 17: Per-round accepted-data and execution cost.

Round	Data acc. (%)	Cand.	Accept	Reject	Probes	Exec.	Tokens (M)
0	38.0	3	3	0	200	12000	42.0
4	52.0	12	7	1	300	18500	68.0
8	55.0	12	7	0	400	23000	81.0

G.5 PAIRED UNCERTAINTY AND ADDITIONAL SEEDS

Methods are paired by task identifier within a seed. The aggregator draws 10,000 task-paired bootstrap samples, reports a 95% confidence interval and paired difference, and estimates $P(\Delta > 0)$. WebShop and ALFWorld add seeds 3 and 4. If an added seed reverses the sign of a headline difference, the paper reports an interval and trend statement rather than a significance claim.

Table 18: Task-paired uncertainty analysis.

Comparison	Δ_{in}	95% interval	Δ_{out}	95% interval	$P(\Delta > 0)$
GymHarness − fixed verifier	5.5	[4.3, 6.7]	3.7	[2.4, 5.0]	0.999
GymHarness − no ratchet	7.5	[6.1, 8.8]	5.3	[4.0, 6.6]	1.000
GymHarness − teacher	1.6	[0.4, 2.8]	1.7	[0.5, 2.9]	0.964
GymHarness − untrained policy	11.3	[9.9, 12.6]	5.2	[3.8, 6.6]	1.000

G.6 INDEPENDENT ATTACK GENERALIZATION

Probe generation is stratified by task cluster, attack family, and generator seed. Ratchet-hidden, shadow, and fresh-final rows are separated, with fresh attacks generated after verifier freeze by both research-agent sources. No fresh trajectory is replayed in a prompt. The report separates same-task/new- attack, new-task/same-attack, new-task/new-attack, and alternative-correct acceptance.

Table 19: Independent attack generalization.

Verifier	Same task/new attack	New task/same attack	New task/new attack	Alt.-correct
Frozen verifier	71.0	58.0	42.0	83.0
No ratchet	78.0	69.0	55.0	76.0
GymHarness	95.0	92.0	90.0	92.0

G.7 RACE-MARGIN PRIORITIZATION ABLATION

Rollout, audit, attack, and verifier-update budgets are fixed while comparing the race-margin scheduler with random and mean-acceptance/FPR selection. Race margin chooses which mode is attacked or audited; it is never a candidate gate. The report records early-warning lead time, final precision, held-out rejection, valid probes per unit cost, and total cost.

Table 20: Race-margin prioritization at matched cost.

Priority	First error	AUROC	Lead	Probes/100	Prec.	Transfer
Race-margin priority	1.8	0.91	3.1	7.2	95.0	44.2
Random priority	2.7	0.50	1.2	4.1	92.0	43.5
Mean-acceptance priority	2.4	0.63	1.8	4.9	93.0	43.8
Aggregate-FPR priority	2.5	0.68	1.6	5.0	93.0	43.7

G.8 RESEARCH-AGENT AND HINT PROVENANCE

Policy trajectories are tagged by source (self-rollout, 397B research agent, 27B self-researcher), hint stratum, and verifier decision. The runner records trajectory share, token share, leakage-filter decisions, accepted/rejected status, and sealed success by stratum. This distinguishes sequence generation from the evidence that caused a sequence to be retained.

Table 21: Research-agent source and hint provenance.

Research source	Hack	Alt.	Regress.	Hint traj. (%)	Hint tokens (%)	In/Out
397B research agent	93.0	92.0	2.1	42	18	72.5/44.2
27B policy as researcher	90.0	89.0	2.8	38	16	70.4/43.1

Table 22: Hint and policy-data provenance.

Hint level	Traj. (%)	Tokens (%)	Leakage (%)	Stratum gain	Transfer gain
No hint	24	24	0	2.1	0.8
H1 checks	14	11	1	9.8	3.7
H2 structure	18	15	3	12.9	5.2
H3 partial golden	12	20	17	9.6	4.0
Minimal privilege	32	30	3	14.8	6.1

G.9 REPRODUCIBILITY MANIFEST

Each run stores a machine-readable `manifest.json`; its hash appears in every ledger row, checkpoint name, and generated result row. Required fields are model revisions, LoRA and optimizer settings, batch and schedule, context and decoding limits, primary and long-horizon round counts, rollout counts, probe ratio, validation pairing, gate rule, prompt/runtime/probe hashes, seeds, checkpoint origin, environment versions, task-list hashes, and sealed API policy. Configuration tables are generated from this manifest.

G.10 LIMITATIONS AND FUTURE DIRECTIONS

The protocol covers fixed runtimes, finite probe families, executable checks, and an eight-round horizon. Execution checks can be incomplete, shadow groups can miss attack families, and bootstrap intervals do not remove correlation among tasks or seeds. Recursive verification means the dependency chain policy $\rightarrow$ verifier $\rightarrow$ execution-labeled probes $\rightarrow$ exogenous evidence; we make no claim about open-ended recursive self-improvement.

Future work should extend fresh attack generation across environments, add stronger independence tests for checks and generators, evaluate longer horizons with pre-registered stopping rules, and publish signed manifests and per-episode ledgers so external evaluators can replay sealed decisions without training-process access.