RRSI: Regularized Recursive Self-Improvement of Agent Harnesses

Peng Xia[1,2*], Rujun Han[1], Zifeng Wang[1], Yanfei Chen[1], Yufan Zhang[1], Yoonho Lee[3], Chengsong Huang[4], Han Yu[1], Zhongying CuiZhu[1], Yifei Ming[1], Huaxiu Yao[2], Burak Gokturk[1], Tomas Pfister[1] and Chen-Yu Lee[1]
[1]Google Google Cloud AI Research, [2] UNC-Chapel Hill, [3] Stanford University, [4] Washington University in St. Louis

An LLM agent's capability is largely magnified by its harness, namely the prompts, control flow, tooling, memory, and context management surrounding the frozen backbone model. Recent methods increasingly automate this process by iteratively proposing and selecting edits of each component of an agent harness, practically establishing a form of recursive self-improvement (RSI) at the agent-system level. However, such recursive evolution may overfit by memorizing the training tasks, showing large in-distribution gains that shrink or even vanish on out-of-distribution benchmarks. We introduce Regularized Recursive Self-Improvement of Agent Harnesses (RRSI), which incorporates the principles of regularizations into harness self-improvement by constraining the evolution candidate proposal and selection. On the proposal side, a budget limits how many edits a candidate can bundle, and this budget is annealed over rounds. The proposer also conditions on the full edit history, so when progress slows, it shifts toward components it has not yet explored instead of continuing to refine the most recent improvement. On the selection side, a candidate must be logically valid for the benchmark, improve performance beyond evaluation noise, and justify any added inference cost with measured gain. A critic performs the benchmark-specific screening, while a noise-adjusted floor, an L_1-style cost rule, and pruning remove changes that are too small, too expensive, or no longer useful. Together these constraints favor reusable agent mechanisms over noisy or benchmark-specific ones. Across eight benchmarks spanning coding (Terminal-Bench, SWE-Bench-Verified), agentic workspace (GDPval, Apex-Agents, JobBench, Harvey-LAB) and engineering design tasks (FrontierEng, EngDesign), RRSI gains up to 14.1 points on the split it evolves against and up to 4.7 points on the five out-of-distribution benchmarks, while producing a harness that runs on a third fewer policy tokens than the unregularized evolution.

github.com/google-research/rrsi | www.rrsi.com

1. Introduction

Modern LLM agents are systems rather than standalone models (Lopopolo, 2026; Rajasekaran, 2026). A frozen backbone model is wrapped in a harness of prompts, control flow, tool interfaces, memory and context management. Agent harness decides whether the same model reads the right file before editing it, recovers from a failed command, manages efficient working context, and writes its findings into the deliverables. Much recent progress in agent products came from harness engineering rather than from new model weights (Karten et al., 2026a; Weng, 2026; Zhang and Khattab, 2026). However, this engineering relies on manual efforts, where humans inspect failed trajectories and tweak the scaffold by hand, so progress is limited by how many trajectories an engineer can read.

Recent methods automate this loop by using LLMs to optimize harness components from task feedback (Chen et al., 2026; Karten et al., 2026b; Lee et al., 2026a,b; Lin et al., 2026a; Lou et al., 2026; Nie et al., 2026; Niklaus, 2026; Zhang et al., 2026a,e). Such iterative harness evolution provides a practical form of recursive self-improvement (RSI) (RSI-Exam Team, 2026; Team et al., 2026; Wang et al., 2025; Zhang et al., 2026b) at the agent-system level, where feedback from the current system is used to improve the harness that shapes its subsequent behavior. However, as illustrated in

* This work was done while Peng was a Student Researcher at Google Cloud AI Research.
Corresponding author(s): pxia@cs.unc.edu, {rujunh, chenyulee}@google.com

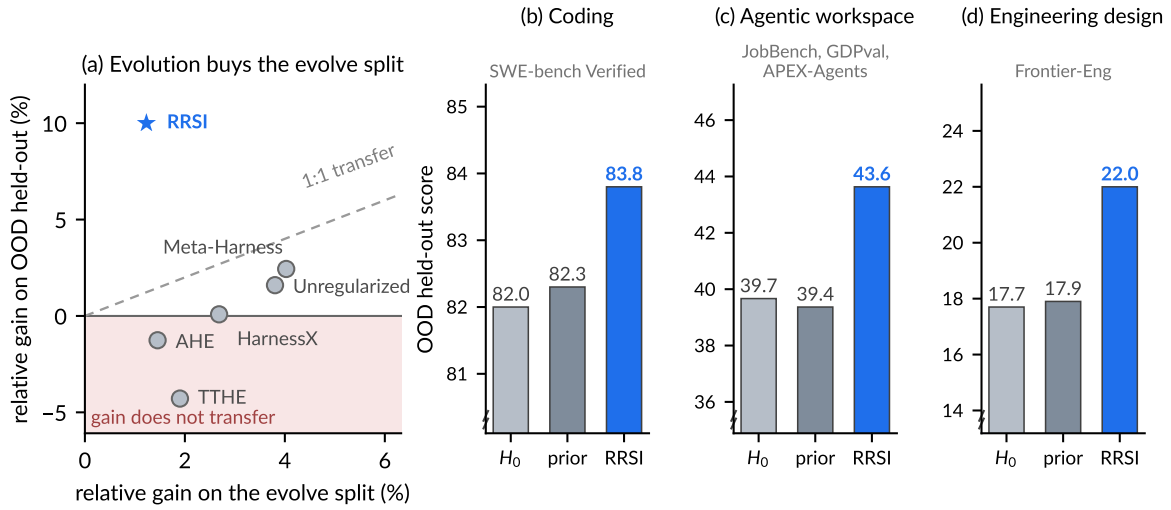

Figure 1 | Evolution overfits on the split it is scored on whereas RRSI generalizes the improvements. (a) Gains on the evolve split against gains out of distribution for the agentic workspace benchmark. Prior methods retain little of their evolve-set gain and several end below H_0, the initial harness. (b-d) Out-of-distribution held-out score for H_0, average of the four baseline methods and our RRSI on SWE-bench Verified, the mean of JobBench, GDPval and APEX-Agents, and Frontier-Eng.

Figure 1 (a), test-time harness evolution repeatedly proposes and selects edits using feedback from a finite evolve set, creating an adaptive overfitting risk: evolve-set performance may improve without corresponding gains on unseen tasks. Recent studies observe substantial gaps between evolution and held-out performance, and show that apparent improvements can arise from task-specific fitting or increased test-time computation rather than reusable mechanisms (Ding et al., 2026; Lin et al., 2026b; Wang et al., 2026b). Accordingly, recent works explicitly separate evolution and evaluation tasks to measure generalization (Huang et al., 2026d; Ke et al., 2026; Zhang et al., 2026d). We therefore view harness evolution for recursive self-improvement as a generalization problem. Here, generalization is defined as transferring to unseen benchmarks with different task descriptions, tool interfaces, or verifiers. In other words, the edits to the harness encode reusable behaviors rather than benchmark-specific shortcuts.

Our study shows that overfitting can arise through several coupled behaviors (Yang et al., 2026a; Zhang et al., 2026d). The evolution search may encode benchmark-specific patterns, promote candidates favored by the evaluation noise, or accumulate complexity that improves evolve-set scores without improving the underlying agent mechanism. These *benchmark-specific fitting*, *noise chasing*, and *complexity accumulation* all widen the evolve-to-transfer gap. Inspired by these observations, our solution regularizes how recursive harness improvements use finite and noisy feedback.

We introduce RRSI, a framework for regularizing the RSI of agent harness that keeps the harness fully editable while constraining how finite evolve-set feedback guides the search. As illustrated in Figure 2, RRSI regularizes both sides of the evolution loop: it encourages simpler and more reusable edits when proposing candidates, and applies robust selection criteria to avoid retaining improvements driven by benchmark-specific signals, evaluation noise, or unnecessary complexity. In this way, RRSI favors edits that transfer beyond evolution set without restricting which harness components may be updated.

We evaluate RRSI on eight benchmarks spanning three domains that differ in task type, tooling and verifier. In each domain the harness is evolved on a single suite, and is then run unchanged on held-out benchmarks. As shown in Figure 1 (b–d), it gains up to 14.1 points on the evolving split and

improves all six held-out splits, by up to 4.7 points out of distribution, on fewer policy tokens than unregularized evolution spends. More importantly, these gains generalize beyond the environment used for evolution. RRSI retains its improvements across substantially different tasks and evaluation settings, indicating that it learns broadly useful harness changes. More importantly, RRSI generalizes across held-out environments, outperforming the average prior baseline by up to 22.9%.

Our contributions are threefold: (1) We identify the overfitting as a key challenge in harness-based recursive self-improvement. (2) We propose RRSI, which regularizes both proposal and selection during harness evolution while keeping each harness component editable. (3) Across eight benchmarks in three domains, RRSI improves both transfer and efficiency, showing the effectiveness of our proposed approach.

2. Preliminaries

Agents and Harnesses. We consider an agent $A = (\pi, H)$ built from a backbone policy π and a harness H. The harness is everything around the weights (Lopopolo, 2026; Rajasekaran, 2026): the system and task prompts, the control flow that decides when the agent plans, acts, reflects or stops, the tool interfaces and their descriptions, the memory and skill files the agent may consult, and the context management that decides what the policy sees at each step. Given a task x with its environment, the agent produces a trajectory $\tau \sim A(\cdot \mid x)$ and a deliverable, which a verifier scores as $r(x, \tau) \in [0, 1]$. The verifier can be a unit-test suite in coding environments or a LLM-as-a-judge program in agentic workspace environments. For a task set $\mathcal{D}$, we measure task performance and policy-token cost as

$$S(H; \mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{\tau\sim A(\cdot|x)}[r(x, \tau)], \qquad C(H; \mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{\tau\sim A(\cdot|x)}[c(\tau)], \tag{1}$$

where $c(\tau)$ is the number of policy tokens consumed by the trajectory.

Harness Evolution. Harness evolution treats H as the optimization variable while keeping the backbone policy fixed (Lee et al., 2026b). Most methods instantiate the same generic loop. At round t, the current harness H_t is executed on an evolve set $\mathcal{D}_{\text{evolve}}$ to obtain trajectories; these trajectories are summarized into feedback $\mathcal{F}_t$; a proposer LLM generates candidate harnesses; the candidates are evaluated on the same evolve set; and the best candidate is selected as the next incumbent. Abstractly,

$$\mathcal{H}_t = \{H_t^{(1)}, \ldots, H_t^{(m_t)}\} \sim P_0(\cdot \mid H_t, \mathcal{F}_t), \qquad H_{t+1} = \operatorname*{arg\,max}_{H' \in \mathcal{H}_t \cup \{H_t\}} \hat{S}(H'; \mathcal{D}_{\text{evolve}}), \tag{2}$$

where P_0 denotes the unconstrained proposal process and $\hat{S}$ is the empirical score obtained from a finite number of stochastic agent runs. With k trials per task, we use

$$\hat{S}(H) = \frac{1}{k|\mathcal{D}_{\text{evolve}}|} \sum_{x\in\mathcal{D}_{\text{evolve}}} \sum_{j=1}^{k} r(x, \tau_x^{(j)}), \qquad \hat{C}(H) = \frac{1}{k|\mathcal{D}_{\text{evolve}}|} \sum_{x\in\mathcal{D}_{\text{evolve}}} \sum_{j=1}^{k} c(\tau_x^{(j)}). \tag{3}$$

Unlike ordinary evaluation, this reuse of $\mathcal{D}_{\text{evolve}}$ is adaptive: the candidates proposed at round t depend on measurements obtained from the same tasks in earlier rounds. Harness evolution can therefore be viewed as adaptive empirical optimization over an unusually expressive search space.

3. RRSI

We study RSI through iterative harness evolution, where feedback from the current agent system is repeatedly used to propose and select modifications to the harness. RRSI follows this recursive improvement process and keeps the harness edit space open, but regularizes how the evolution moves

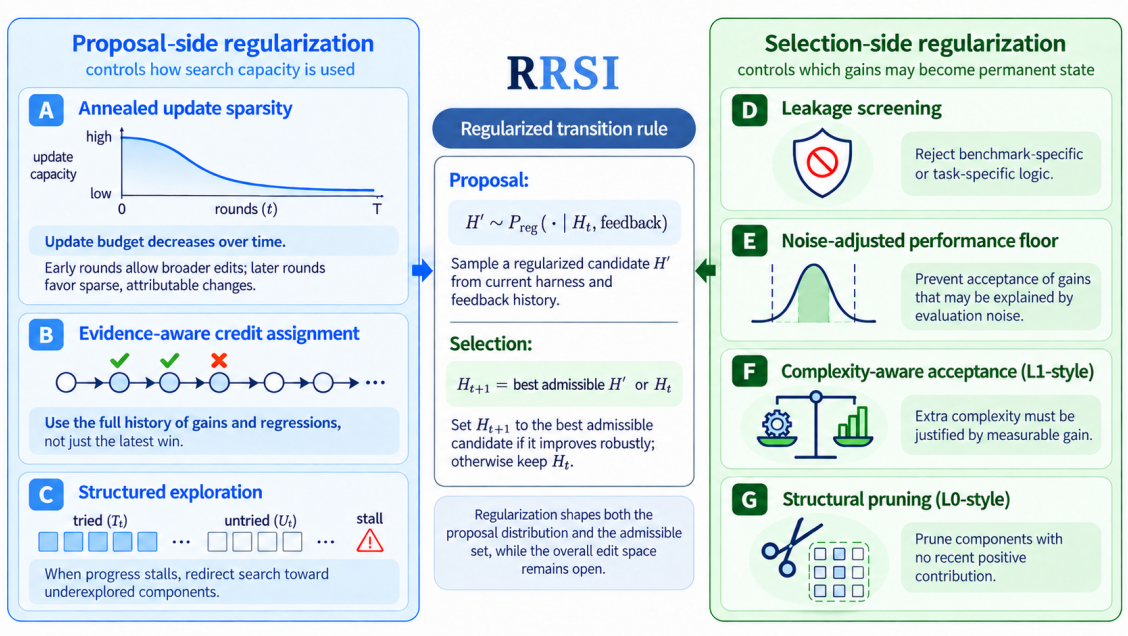

Figure 2 | Overview of RRSI. RRSI regularizes the search trajectory, not restricting the potential harness edit space: proposal-side constraints control how search capacity is used, while selection-side constraints control which measured improvements are allowed to become a permanent state.

through that space. The key idea is to translate regularization principles from machine learning into an adaptive harness search: sparse updates limit how many mechanisms can change in response to one round of feedback, evidence-aware credit assignment prevents the search from repeatedly spending its capacity on hypotheses it has already falsified, and conservative selection prevents leakage, evaluation noise, or unjustified resource growth from becoming permanent harness state.

3.1. A Regularization View of Harness Evolution

Let $\Omega(H)$ denote the set of harnesses reachable from H by arbitrary source edits. RRSI deliberately leaves $\Omega(H)$ open: prompts, control flow, configuration, context management, tools, skills, memory, and subagents may all be modified, added, or removed. Instead of restricting this hypothesis space directly, we regularize the search trajectory through it. At each round t, the proposer uses feedback from the finite evolve set to generate candidate edits to the current harness H_t, and the selector determines which, if any, should replace the incumbent.

This view separates two complementary forms of regularization. On the proposal side, we constrain how much adaptive capacity can be exercised in a single round and where that capacity is spent. On the selection side, we constrain which empirical improvements are strong enough, efficient enough, and sufficiently free of leakage to survive. Our method draws on two standard regularization principles. On the proposal side, we limit the number of independent edits in each candidate, which is analogous to an L_0 sparsity constraint that favors simpler updates. On the selection side, we penalize aggregate inference growth, which plays a role analogous to an L_1-style magnitude penalty that discourages unnecessary complexity (Goodfellow et al., 2016; Hastie et al., 2009; Louizos et al., 2018). Detailed algorithm description can be found in Appendix B.

3.2. Regularizing the Proposal Distribution

The proposal distribution determines how aggressively the search can respond to feedback from the evolve set. RRSI regularizes it in three ways: it anneals how much update capacity a single round may exercise, it makes credit assignment evidence-aware over the whole run, and it structures where that capacity is spent.

Annealed Update Sparsity. An unconstrained proposer can bundle many unrelated modifications into one candidate. Such candidates have high effective capacity: they can fit more idiosyncrasies of the current feedback, and any measured change is difficult to attribute to a particular mechanism. We therefore cap the number of independently attributable edits that may be included in one proposal. At round t of a T-round run, this budget is

$$b_t = \left\lceil b_{\min} + (b_{\max} - b_{\min}) \cdot \tfrac{1}{2}\big(1 + \cos(\pi t/T)\big) \right\rceil. \tag{4}$$

The schedule decreases from $b_{\max}$ to $b_{\min}$: early rounds may combine several coordinated changes to discover new mechanisms, whereas later rounds become increasingly sparse and attributable. This annealing acts as a form of regularization on the update process: early rounds allow broader changes, while later rounds progressively restrict the search to smaller, more attributable edits.

Evidence-Aware Credit Assignment. Constraining the size of an update only helps if the search knows what earlier updates established. Every evaluation is another adaptive look at the same finite evolve set, so repeatedly testing hypotheses that earlier rounds already falsified spends search capacity without adding useful evidence (Dwork et al., 2015). RRSI therefore records, for every evaluated candidate, the component it modifies, the hypothesis it tests, the source diff, the resulting score and cost changes, and whether the candidate was accepted. The proposer conditions on this history in later rounds: rejected mechanisms remain negative evidence, while successful mechanisms retain explicit credit. As later rounds allow fewer edits per candidate, it becomes easier to identify which change is responsible for an observed improvement.

Structured Exploration. The same history reveals when the proposer has collapsed onto a narrow edit family, for example repeatedly rewriting prompts while leaving agent structural mechanisms untouched. We treat the search as stalled when its progress over the previous w rounds remains within the empirical noise band δ. During a stall, a small portion of the proposal budget is reserved for components that have not yet been exercised in the run. This plays a role similar to diversity or entropy regularization: it redirects limited proposal capacity toward underexplored mechanisms without changing which mechanisms the harness is allowed to contain (Haarnoja et al., 2018).

3.3. Regularizing Candidate Selection

Standard harness evolution can promote the candidate with the largest measured score even when that score reflects explicit leakage, stochastic variation, or costly growth. RRSI retains the same empirical objective but regularizes which candidates are allowed to become permanent state. A candidate must satisfy several non-compensatory criteria before its score can justify replacing the incumbent.

Leakage Screening. Before full evaluation, a critic reads each candidate diff and rejects edits that explicitly encode task names, entity names, task-specific values, answers, or other logic specific to the

evolve benchmark, as well as edits that add inert machinery. The screen targets benchmark-specific content rather than particular harness components: generic prompt or tool-description improvements remain valid candidates. Screening before evaluation is important because a leaking candidate never receives the inflated evolve-set score that could make it attractive to subsequent rounds.

Stability-Aware Acceptance. Repeatedly selecting among noisy evaluations can convert stochastic winners into permanent search state. Before evolution, we repeatedly evaluate the unchanged base harness and estimate an empirical noise band δ. Let $S^\star$ denote the best evolve-set score observed so far. A candidate must satisfy the noise-adjusted floor

$$\hat{S}(H') \geq S^\star - \delta. \tag{5}$$

The floor prevents the search from walking downhill through a sequence of regressions that are individually small enough to be mistaken for noise. More broadly, it makes selection conservative to fluctuations induced by repeated stochastic evaluation on the same evolve set (Dwork et al., 2015).

L_1-Style Complexity-Aware Acceptance. For a candidate H' relative to the current harness H_t, let

$$\Delta S = \hat{S}(H') - \hat{S}(H_t), \qquad \Delta C = \frac{\hat{C}(H') - \hat{C}(H_t)}{\hat{C}(H_t)}. \tag{6}$$

For a candidate whose gain exceeds the noise band, $\Delta S > \delta$, we require

$$\Delta C \leq \beta_0 + \beta_1 \Delta S. \tag{7}$$

Here, β_0 sets the cost increase tolerated for a negligible score gain, while β_1 controls how much additional cost is allowed as the measured improvement increases. We select these values on the evolve set and keep them fixed for all transfer evaluations. Thus additional inference cost must be justified by measurable performance improvement. This plays a role similar to L_1 regularization: rather than penalizing the number of active structures, it discourages excessive aggregate magnitude, here represented by the resource footprint of the evolved harness. We use policy-token cost as a common measurable proxy for this footprint. The detailed rule for candidates whose measured change falls within the noise band is deferred to the Appendix B.

L_0-Style Structural Pruning. The annealed budget in Equation (4) sparsifies each *update*; pruning sparsifies the *retained harness*. RRSI tracks whether recently exercised components have produced a strictly positive measured gain over a fixed pruning window. Components that remain unproductive are reported to the proposer as deletion targets in subsequent rounds. This plays a role analogous to L_0-style structural sparsification: a mechanism must continue to earn its place rather than persist simply because score-only evolution has no incentive to remove it (Louizos et al., 2018).

4. Experiments

We evaluate RRSI on eight benchmarks spanning three domains: Terminal-Bench 2.1 and SWE-bench Verified for coding, Harvey LAB, JobBench, GDPval and APEX-Agents for agentic workspace tasks, and EngDesign and Frontier-Eng for engineering design. Our experiments address the following questions: 1) How does RRSI compare to state-of-the-art harness evolution methods? 2) Do the gains transfer to in-distribution held-out tasks and to out-of-distribution benchmarks that the search never saw? 3) What is the contribution of different components? 4) How does the harness evolve over a run, and what does it cost in tokens?

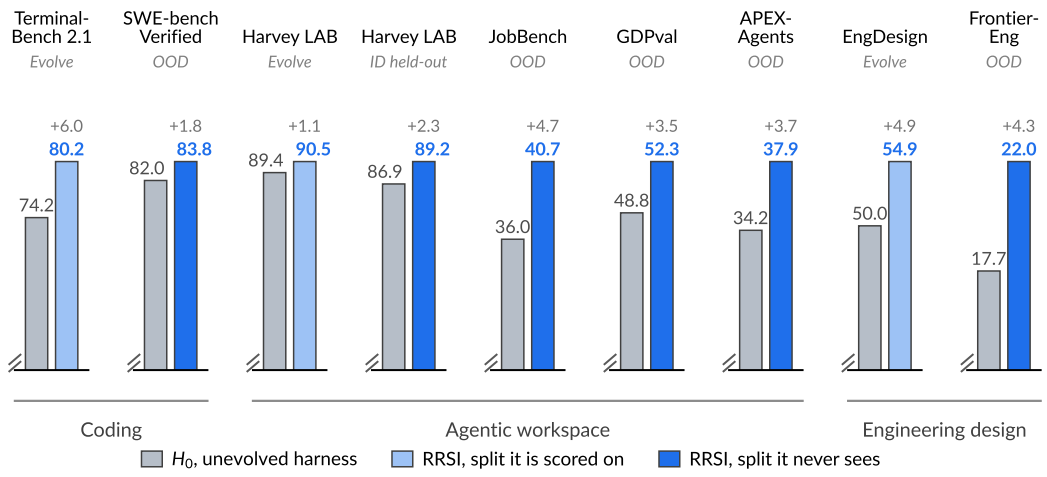

Figure 3 | Main results in all three domains.

4.1. Experimental Setup

Environments. We evolve harnesses in three settings that differ in task type, i.e., coding tasks, agentic workspace tasks and engineering design tasks. For coding, Terminal-Bench 2.1 (Merrill et al., 2026) is a suite of 89 containerized terminal tasks in which the agent drives a real shell and is verified by the task's own unit tests. For agentic workspace tasks, Harvey LAB (Harvey AI, 2026) is a legal-work benchmark spanning 25 practice areas. Harvey LAB is split into a fixed evolve set of 120 tasks and a pristine in-distribution held-out set of 40 tasks. For engineering design, EngDesign (Guo et al., 2025) contributes 61 design tasks, each graded by its own frozen simulator rather than by a judge model. To test its generalization capability, we additionally evaluate on out-of-distribution (OOD) held-out benchmarks, e.g., SWE-bench Verified (Jimenez et al., 2024) for repository-level bug fixing on the coding instance, JobBench (Li et al., 2026), GDPval (Patwardhan et al., 2026) and APEX-Agents (Vidgen et al., 2026) on the agentic workspace instance, and Frontier-Eng (Chi et al., 2026) on the engineering design instance.

Baselines. We compare against the unevolved base harness H_0 that every run starts from, and against four recent harness evolution methods, Meta-Harness (Lee et al., 2026b), AHE (Lin et al., 2026a), TTHE (Nie et al., 2026) and HarnessX (Chen et al., 2026). All the baselines start from the same H_0 and share the frozen policy, the evolve set and the candidate budget.

Implementation Details. The policy is frozen throughout Claude Opus 4.8 (Anthropic, 2026a) across all three domains. The proposer, the analyst that writes the cross-round failure feedback and the leakage critic are all Claude Opus 4.8. The base harness we used are Terminus-2 (for coding) (Merrill et al., 2026), a ReAct loop (Yao et al., 2022) over an MCP tool gateway, a dynamic toolbelt (Vidgen et al., 2026), and ReSum-style context management (for Harvey LAB and EngDesign). Further hyperparameters are given in Appendix C.1.

4.2. Main Results

RRSI improves every split outside the evolve set, in all three domains. Figure 3 reports every number against the unevolved base harness H_0 measured in the same window, so no gain can be attributed to drift in the evaluation infrastructure. The evolve-set gains are 6.0 points on Terminal-Bench 2.1, 4.9 on EngDesign and 1.1 on Harvey LAB. What matters is what remains once the harness leaves those splits. SWE-bench Verified gains 1.8 points although repository-level bug fixing was never

Method	In-Distribution		Out-of-Distribution		
	Harvey LAB (Evolve)	Harvey LAB (ID Held-out)	JobBench	GDPval	APEX-Agents
H_0 (no evolution)	89.4	86.9	36.0	48.8	34.2
Meta-Harness (Lee et al., 2026b)	**93.0**	89.2	37.1	49.1	35.7
AHE (Lin et al., 2026a)	90.7	88.7	37.2	47.2	33.1
TTHE (Nie et al., 2026)	91.1	88.5	35.2	47.0	31.7
HarnessX (Chen et al., 2026)	91.8	89.1	36.3	48.5	34.3
RRSI (ours)	90.5	**89.2**	**40.7**	**52.3**	**37.9**

Table 1 | Comparison with prior harness evolution methods on agentic workspace tasks.

scored. The in-distribution held-out split of Harvey LAB gains 2.3, and the three out-of-distribution agentic benchmarks gain between 3.5 and 4.7 points, 7.2% to 13.1%. Frontier-Eng gains 4.3 Medal points, a 24.3% relative improvement. No held-out split regresses anywhere, which is the failure a memorizing harness produces.

RRSI consistently outperforms baselines on all held-out datasets. Table 1 runs the four prior methods from the same H_0 on the same evolve split under the same candidate budget. Every one of them works well on evolve set. The performance on in-distribution held-out split is quite similar. The separation appears out of distribution, and there the ranking inverts. Meta-Harness, the strongest baseline on the evolve split, adds 0.9 points to the out-of-distribution average; HarnessX lands on the base one; AHE and TTHE finish below the harness they started from, TTHE by 1.7 points. RRSI posts the smallest evolve-set gain of any evolved harness and the only out-of-distribution average that clears H_0 by more than a point, 43.6 against 39.7, which is the trade the regularizers are designed to make.

The transfer is not an artifact of judge-mediated grading or of a shared task format. Harvey LAB, JobBench and GDPval are all scored by a judging model, so a harness could in principle raise its score by writing the way a judge rewards rather than by producing better work. The engineering design instance closes that route: each EngDesign and Frontier-Eng task is graded by its own simulation or testbench, the grading is deterministic, and a design either meets the stated constraints or does not. The gains survive there unchanged, and deterministic grading also removes judge variance from the measurement.

4.3. Analysis

The main results establish that the evolved harnesses transfer; this section asks what produced that property, whether it depends on the backbone the search was run with, and what it costs. Unless stated otherwise, every run below uses the agentic workspace instance and shares the base harness, policy, evolve split, round count and candidate budget of the main experiment, so that arms differ only in the factor under study.

Ablation Analysis. We ablate the two groups of regularizers, (i) the proposal-side constraints and (ii) the acceptance-side constraints. As shown in Table 2, removing either group raises the evolve-set score and lowers transfer. Without the acceptance constraints the evolve-set score rises from 90.5 to 91.5 while the out-of-distribution average falls from 43.6 to 41.0 and token cost rises by half, showing that an unconstrained selection rule spends most of its accepted edits on noise and on context rather than on mechanism. Removing the proposal constraints costs only 0.2 points on the evolve split but 1.7 out of distribution, suggesting that steering where the search looks matters even when nothing is rejected. The most significant degradation comes from removing both, which lifts the evolve-set score to 92.8, the highest of any arm, and leaves the out-of-distribution average at 40.3, within a point of

Variant	Harvey LAB (Evolve)	Harvey LAB (ID Held-out)	OOD Avg.	Tokens/trial (m) ↓
H_0 (no evolution)	89.4	86.9	39.7	1.56
Unregularized evolution	92.8	88.9	40.3	3.80
w/o proposal regularizers	90.7	88.8	41.9	2.69
w/o acceptance regularizers	91.5	88.7	41.0	3.59
RRSI	90.5	89.2	43.6	2.42

Table 2 | Ablation study of the regularizers on agentic workspace tasks. OOD Avg. is the mean over JobBench, GDPval and APEX-Agents.

Policy	Benchmark	H_0	RRSI	Δ
Claude Opus 4.8	Terminal-Bench 2.1 (Evolve)	74.2	**80.2**	+6.0
	SWE-bench Verified (OOD)	82.0	**83.8**	+1.8
Gemini 3.5 Flash	Terminal-Bench 2.1 (Evolve)	64.6	**78.7**	+14.1
	SWE-bench Verified (OOD)	76.8	**79.0**	+2.2

Table 3 | Policy robustness in the coding domain. Harness evolution is run independently with each frozen policy on Terminal-Bench 2.1, and the resulting harness is evaluated unchanged on SWE-bench Verified.

the unevolved harness, at 3.80 million tokens per trial against our 2.42.

RRSI is not tied to one policy family. To test whether the gains from regularized harness evolution depend on the policy used during search, we independently run the coding evolution with two policy models from different families: Claude Opus 4.8 and Gemini 3.5 Flash (Google, 2026b). For each policy, we start from the same coding harness, evolve only on Terminal-Bench 2.1, and evaluate the resulting harness on both the evolve benchmark and SWE-bench Verified. As shown in Table 3, under Gemini 3.5 Flash, RRSI improves Terminal-Bench 2.1 from 64.6 to 78.7 and transfers a 2.2-point gain to SWE-bench Verified. Under Claude Opus 4.8 the pattern is the same: Terminal-Bench 2.1 rises from 74.2 to 80.2 and SWE-bench Verified from 82.0 to 83.8, although the stronger policy starts closer to the ceiling of both suites and leaves less room to gain. In both cases the harness improves the unseen benchmark without ever being scored on it, which suggests that the benefits of RRSI are not specific to a particular backbone.

The evolved harness still helps under a backbone the search never used. A harness is a program, not a set of weights, so a mechanism that helps only the policy it was searched against is an artifact of that policy rather than a reusable one. We take the final harness of the coding run, evolved with Gemini 3.5 Flash, and evaluate it unchanged with Gemini 3.1 Flash Lite, a smaller model that never took part in the search. As shown in Table 4, Terminal-Bench 2.1 accuracy rises from 11.2 to 14.6, a 30.4% relative gain against a base score less than a fifth of the search policy's. The mechanisms therefore do not depend on the capability level they were searched at, although the absolute gain is smaller because a weaker backbone leaves fewer tasks within reach of any harness.

RRSI produces the lightest harness of any evolved harness. Two regularizers act directly on cost: the L_1-style budget refuses growth that is not paid for when it is proposed, and the pruning rule removes growth that has stopped being paid for since. No prior method carries either constraint, and Figure 4 (a) shows the consequence: all four sit in the region RRSI dominates, spending more policy tokens per trial for a lower out-of-distribution average. AHE is the extreme case, at 3.82 million tokens per trial, 58% more than ours, for 4.4 points less out of distribution. The ordering carries over to trajectory length in Figure 4 (b), where RRSI runs 26.3 steps per trial against 27.3 to 34.6 for

Evaluation policy	H_0	RRSI	Δ
Gemini 3.5 Flash (search policy)	64.6	**78.7**	+14.1
Gemini 3.1 Flash Lite (unseen)	11.2	**14.6**	+3.4

Table 4 | Cross-model transfer on Terminal-Bench 2.1. The harness evolved with Gemini 3.5 Flash as the frozen policy is run unchanged with a weaker backbone that never took part in the search.

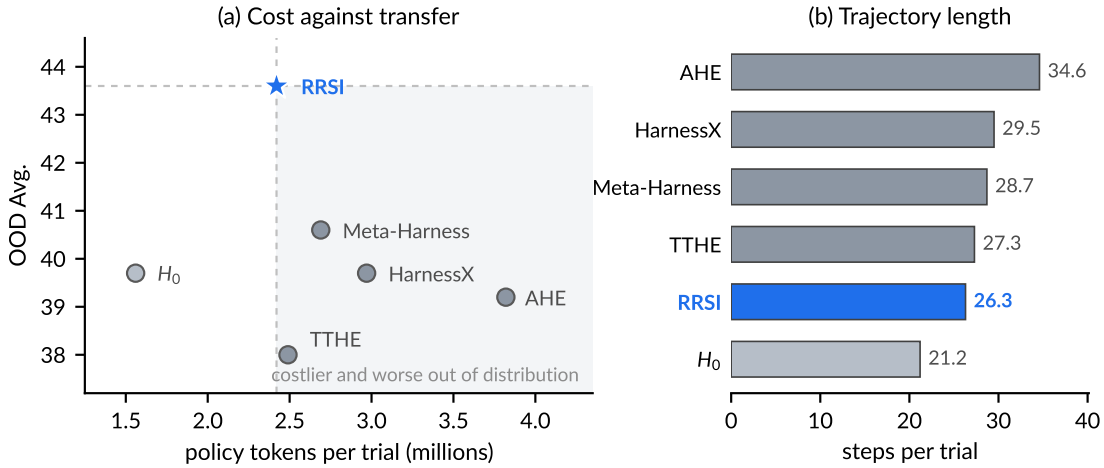

Figure 4 | Cost of the final harness of each arm, measured on the evolve split of the agentic workspace instance. OOD Avg. is the mean over JobBench, GDPval and APEX-Agents. The shaded region in (a) is everything RRSI dominates: more policy tokens per trial for a lower out-of-distribution average.

the prior methods. No evolved harness is as cheap as H_0, at 1.56 million tokens and 21.2 steps, so evolution does buy part of its gain with test-time compute; the budget decides how much.

5. Related Work

Agent harnesses. The harness, and not only the backbone model, determines what an agent can accomplish: engineering reports from frontier labs describe how prompt structure, tool interfaces, context compaction and recovery logic decide whether a long-running agent finishes a task at all (Lopopolo, 2026; Rajasekaran, 2026), and recent analyses argue that harnesses compose and generalize in their own right (Wang et al., 2026a; Weng, 2026; Zhang and Khattab, 2026). A well-designed harness can even substitute for scale, recovering much of a larger backbone's capability at a fraction of the cost (Yang et al., 2026a). This engineering is overwhelmingly manual, and because the best harness is tied to a specific backbone, its cost is paid again with every model release (Huang et al., 2026b).

Harness evolution. The closest line of work automates that loop: an LLM proposer rewrites the harness and edits are kept if they raise a benchmark score (Chen et al., 2026; Karten et al., 2026b; Lee et al., 2026a,b; Lin et al., 2026a; Liu et al., 2026b; Nie et al., 2026; Zhang et al., 2026a), or a single component is evolved, such as skills (Xia et al., 2026a; Yang et al., 2026b), memory (Liu et al., 2026a; Ouyang et al., 2026; Tang et al., 2025; Wu et al., 2026) or a preference signal over rollouts (Pan et al., 2026). This inherits both the mechanisms and the risks of self-improving agents that search over their own code under an empirical fitness signal (Huang et al., 2026a,c; Wang et al., 2025; Xia et al., 2026b,c; Zhang et al., 2026b,c). Throughout, the search is driven by the score on the suite it optimizes against, with no term for generalization, and the cost is not hypothetical: reported gains

often do not survive a change of suite (Huang et al., 2026d; Wang et al., 2026b), and delta attribution separates edits that install a reusable mechanism from those that merely fit the evolution tasks (Ding et al., 2026). Concurrent work also targets generalization directly, either as an explicit objective of the search (Zhang et al., 2026d) or by replacing greedy selection with a diversity-preserving archive over candidate harnesses (Luo et al., 2026). Our contribution is orthogonal to what these methods edit. We keep the same open edit space and instead regularize the search dynamics: credit assigned over the full evolution history, task-specific logic filtered before scoring, and acceptance against a noise-adjusted baseline, so what survives is a mechanism rather than a fit to the evolution suite.

6. Conclusion

We study iterative harness evolution as a practical form of recursive self-improvement at the agent-system level, and show that this recursive process itself requires regularization. Because a finite evolve set is reused adaptively across rounds, apparent self-improvement can reflect benchmark-specific fitting, evaluation noise, or unnecessary complexity rather than transferable progress. RRSI addresses this problem by regularizing both proposal and selection while leaving the harness edit space open. Across coding, agentic workspace, and engineering design tasks, the resulting harnesses improve held-out and cross-benchmark performance while using less inference cost than unregularized evolution. These results suggest that making agent systems increasingly capable through recursive self-improvement requires controlling not only what can change, but also how repeated feedback is converted into persistent changes.

Limitations

XXXX

References

Anthropic. Introducing claude opus 4.8, 2026a. URL https://www.anthropic.com/news/claude-opus-4-8.

Anthropic. Introducing claude sonnet 4.6, 2026b. URL https://www.anthropic.com/news/claude-sonnet-4-6.

T. Chen, S. Lu, K. Zhao, W. Meng, H. Teng, T. Li, C. Li, X. Liu, J. Liang, Z. Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

Y. Chi, D. Hong, D. Jiang, T. Luo, K. Yang, B. Zhang, Z. Cao, X. Fan, B. He, H. Hao, et al. Frontier-eng: Benchmarking self-evolving agents on real-world engineering tasks with generative optimization. *arXiv preprint arXiv:2604.12290*, 2026.

W. Ding, Q. Lu, C. Yu, S. Li, S. Jin, X. Liu, and G. Durrett. What evolves when we talk about harness evolution? *wenwen-d.github.io*, August 2026. URL https://wenwen-d.github.io/blog/harness-delta-attribution/.

C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth. Generalization in adaptive data analysis and holdout reuse. *Advances in neural information processing systems*, 28, 2015.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Google. Gemini 3.1 pro: Best for complex tasks and bringing creative concepts to life, 2026a. https://deepmind.google/models/gemini/pro/.

Google. Gemini 3.5: frontier intelligence with action, 2026b. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-5/.

X. Guo, Y. Li, X. Kong, Y. Jiang, X. Zhao, Z. Gong, Y. Zhang, D. Li, T. Sang, B. Zhu, et al. Toward engineering agi: Benchmarking the engineering design capabilities of llms. *Advances in Neural Information Processing Systems*, 2025.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

Harvey AI. Harvey lab: The legal agent benchmark, 2026. URL https://github.com/harveyai/harvey-labs/tree/v1.0. Announcement: https://www.harvey.ai/blog/introducing-harveys-legal-agent-benchmark.

T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

C. Huang, H. Liu, T. Zheng, R. Dai, L. Huang, J. Li, Z. Li, Z. Wei, Y. Meng, and J. Huang. G-zero: Self-play for open-ended generation from zero data. *arXiv preprint arXiv:2605.09959*, 2026a.

C. Huang, Z. Wang, R. Han, J. Yan, Y. Chen, Z. CuiZhu, K. Jiang, P. Xia, H. Yu, Y. Zhuang, et al. Envharness: Awakening static worlds for agent learning. *arXiv preprint arXiv:2608.19880*, 2026b.

C. Huang, W. Yu, X. Wang, H. Zhang, Z. Li, R. Li, J. Huang, H. Mi, and D. Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pages 130770–130790, 2026c.

L. Huang, C. Yang, H. Zhou, H. Song, Z. Chen, R. Le, Y. Song, W. X. Zhao, and T. Zhang. Evo-bench: Can language models improve agent harness? *arXiv preprint arXiv:2608.09096*, 2026d.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

S. Karten, A. L. Zhang, K. Thomas, S. Müller, and P. I. Team. Prime agent: A self-improving rlm harness. *Prime Intellect Blog*, 2026a.

S. Karten, J. Zhang, T. Upaa Jr, R. Feng, W. Li, C. Shi, C. Jin, and K. Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026b.

Z. Ke, V. Patil, H. Shi, Y. Li, Y. Liu, S. Shekkizhar, A. Koul, J. Wang, X. P. Nguyen, S. Yavuz, et al. Evoharnessbench: Can your agents keep pace with an evolving harness? *arXiv preprint arXiv:2609.04280*, 2026.

H. Lee, J. Xu, J. Seely, D. Lee, M. Zaharia, and Y. Tang. Recursive harness self-improvement. *arXiv preprint arXiv:2607.15524*, 2026a.

Y. Lee, R. Nair, Q. Zhang, K. Lee, O. Khattab, and C. Finn. Meta-harness: End-to-end optimization of model harnesses. *The Third Conference on Language Modeling*, 2026b.

Y. Li, Y. Feng, Z. Xu, Z. Ma, K. Zheng, F. Jiang, X. Sun, R. Shao, Z. Chen, Y. Huang, et al. Jobbench: Aligning agent work with human will. *arXiv preprint arXiv:2605.26329*, 2026.

J. Lin, S. Liu, C. Pan, L. Lin, S. Dou, Z. Xi, X. Huang, H. Yan, Z. Han, T. Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026a.

M. Lin, J. Wu, Z. Wang, Z. Shi, Y. Sang, B. He, Z. Liu, T. Wei, Z. Wu, Z. Zhang, et al. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. *arXiv preprint arXiv:2605.30621*, 2026b.

J. Liu, X. Ye, P. Xia, Z. Zheng, C. Xie, M. Ding, and H. Yao. Evolvemem: Self-evolving memory architecture via autoresearch for llm agents. *arXiv preprint arXiv:2605.13941*, 2026a.

Z. Liu, Z. Shi, Y. Sang, B. He, M. Lin, T. Wei, D. Wang, B. Dumoulin, W. Jin, and H. Lu. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams. *arXiv preprint arXiv:2606.01770*, 2026b.

R. Lopopolo. Harness engineering: leveraging codex in an agent-first world, 2026. https://openai.com/index/harness-engineering/.

X. Lou, M. Lázaro-Gredilla, A. Dedieu, C. Wendelken, W. Lehrach, and K. P. Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

X. Luo, F. Wang, C. Hu, D. Xue, and Y. Deng. Self-evolving agent harnesses via gated semantic quality-diversity. *arXiv preprint arXiv:2607.13683*, 2026.

M. Merrill, A. Shaw, N. Carlini, B. Li, H. Raj, I. Bercovich, L. Shi, J. Shin, T. Walshe, E. K. Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pages 40903–40986, 2026.

J. Nie, Y. Zhang, J. Song, Q. Cai, D. Yu, Y. Guo, X. Tian, and B. Han. Tthe: Test-time harness evolution. *arXiv preprint arXiv:2607.08124*, 2026.

J. Niklaus. Don't train the model, evolve the harness, 2026. URL https://huggingface.co/spaces/joelniklaus/harness-optimization.

S. Ouyang, J. Yan, I. Hsu, Y. Chen, K. Jiang, Z. Wang, R. Han, L. Le, S. Daruki, X. Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pages 94327–94354, 2026.

W. Pan, S. Liu, C.-Y. Lin, J. Zeng, X. Tang, X. Zhou, Y. Lu, and X. Jia. Retrospective harness optimization: Improving llm agents via self-preference over trajectory rollouts. *arXiv preprint arXiv:2606.05922*, 2026.

T. Patwardhan, R. Dias, E. Proehl, G. Kim, M. Wang, O. Watkins, S. Fishman, M. Aljubeh, P. Thacker, L. Fauconnet, et al. Gdpval: Evaluating ai model performance on real-world economically valuable tasks. In *International Conference on Learning Representations*, volume 2026, pages 24005–24040, 2026.

Qwen Team. Qwen3.6-Plus: Towards real world agents, April 2026. URL https://qwen.ai/blog?id=qwen3.6.

P. Rajasekaran. Harness design for long-running application development, 2026. https://www.anthropic.com/engineering/harness-design-long-running-apps.

RSI-Exam Team. Rsi-exam: Benchmarking recursive self-improvement through executable research, 2026. URL https://github.com/aiming-lab/RSI-Exam.

X. Tang, T. Qin, T. Peng, Z. Zhou, D. Shao, T. Du, X. Wei, P. Xia, F. Wu, H. Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

N. Team, G. Cao, G. Dai, T. Guo, K. Han, H. Hu, Z. Jiang, X. Kuang, B. Li, Y. Li, et al. Neohorse-1: Towards recursive self-improvement via agentic post-training with routing harness. *arXiv preprint arXiv:2609.08183*, 2026.

B. Vidgen, A. Mann, A. Fennelly, J. W. Stanly, L. Rothman, M. Burstein, J. Benchek, D. Ostrofsky, A. Ravichandran, D. Sur, et al. Apex-agents. *arXiv preprint arXiv:2601.14242*, 2026.

R. Wang, Y. Shi, Z. Li, Z. Li, Y. Yu, J. Yang, K. Panaganti, H. Mi, D. Zhou, et al. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026a.

W. Wang, P. Piękos, L. Nanbo, F. Laakom, Y. Chen, M. Ostaszewski, M. Zhuge, and J. Schmidhuber. Huxley-g\" odel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. *arXiv preprint arXiv:2510.21614*, 2025.

Y. Wang, H. Zhu, Z. Hu, Y. Yuan, Z. Chen, S. Senthil, H. Hajishirzi, Y. Tsvetkov, P. Dasigi, and T. Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026b.

L. Weng. Harness engineering for self-improvement. *lilianweng.github.io*, July 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/.

S. Wu, H. Zhu, Y. Zhang, X. Wang, and S. Yeung-Levy. Automem: Automated learning of memory as a cognitive skill. *arXiv preprint arXiv:2607.01224*, 2026.

P. Xia, J. Chen, H. Wang, J. Liu, K. Zeng, Y. Wang, S. Han, Y. Zhou, X. Zhao, H. Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026a.

P. Xia, J. Chen, X. Yang, H. Tu, J. Liu, K. Xiong, S. Han, S. Qiu, H. Ji, Y. Zhou, et al. Metaclaw: Just talk–an agent that meta-learns and evolves in the wild. *arXiv preprint arXiv:2603.17187*, 2026b.

P. Xia, K. Zeng, J. Liu, C. Qin, F. Wu, Y. Zhou, C. Xiong, and H. Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *The Third Conference on Language Modeling*, 2026c.

C. Yang, X. Zhao, T. Wu, and C. Kästner. Better harnesses, smaller models: Building 90% cheaper agents via automated harness adaptation. *arXiv preprint arXiv:2607.08938*, 2026a.

Y. Yang, Z. Gong, W. Huang, Q. Yang, Z. Zhou, Z. Huang, Y. Li, X. Gao, Q. Dai, B. Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026b.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

A. Zhang and O. Khattab. Language model harnesses are compositional generalizers. July 2026. URL https://alexzhang13.github.io/blog/2026/harness/.

H. Zhang, S. Zhang, K. Li, C. Zhang, Y. Chen, Y. Zhang, L. Bai, and S. Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a.

J. Zhang, S. Hu, C. Lu, R. Lange, and J. Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pages 104223–104294, 2026b.

J. Zhang, B. Zhao, W. Yang, J. Foerster, J. Clune, M. Jiang, S. Devlin, and T. Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026c.

L. Zhang, R. Zhou, D. Song, Z. Chen, Y. Tian, J. Yang, H. Ma, C. Li, G. Feng, X. Li, et al. Harnesscompass: Guiding automatic harness evolution toward generalizable and effective agent harnesses. *arXiv preprint arXiv:2608.01918*, 2026d.

Y. Zhang, Y. Dai, J. Tan, L. Yang, R. Mullur, T. Hoang, Z. Hu, J. Zhu, P. Mui, S. Savarese, et al. Darwinx: Evolving agent harnesses through natural selection. *arXiv preprint arXiv:2608.07545*, 2026e.

Contents of Appendix

A. Evaluation

This shows how each environment is run and scored. A harness and its baseline are always evaluated in the same window, with the same tool environment, the same judge and the same number of trials.

A.1. Terminal-Bench 2.1

Each task is a container image with a task description, a working directory and a set of unit tests that are hidden from the agent. The agent drives a real shell through the harness, and a task counts as solved only if the task's own test suite passes after the agent stops, so the reward is exact and cannot be produced by a plausible-looking answer. The reported accuracy is the fraction of the 89 tasks solved in this way. Containers are torn down and rebuilt between arms so that no state carries from one evaluation to the next.

A.2. SWE-bench Verified

Each instance is a real GitHub issue paired with the repository snapshot at the time of the report. The agent must produce a patch, which is then applied to the snapshot and checked against the instance's fail-to-pass tests, which must go from failing to passing, and its pass-to-pass tests, which must remain passing. The reported resolve rate is the fraction of instances that satisfy both conditions.

A.3. Harvey LAB

Each task provides a folder of source documents in Word, Excel and PDF form and requires the agent to produce deliverable files under exact requested filenames, which are graded by a strict per-criterion rubric of 20 to 100 independently judged criteria per task, roughly 14,000 criterion verdicts per full evaluation. A criterion is judged in isolation by an LLM judge (Gemini-3.5-Flash (Google, 2026b)) that reads the produced deliverable together with that single criterion, and the score of a run is the fraction of criteria passed over all tasks, so a task with a long rubric contributes proportionally more evidence than a short one and a missing deliverable fails every criterion it was supposed to satisfy rather than being dropped. The 160 tasks are partitioned once into a 120-task evolve set and a 40-task held-out set, and the partition is fixed for the experiment.

A.4. JobBench

Tasks are drawn from real professional workflows, each shipping a task folder of input files and a wrapper prompt, with the reference material the agent would need to look up deliberately withheld so that part of the work is genuine retrieval. The harness exposes a filesystem, a code execution tool for producing office and PDF deliverables, and a grounded web search tool. Deliverables are graded by the benchmark's own weighted rubric, and the reported number is the weighted rubric score over the evaluated split. We use an LLM judge (average score of Gemini-3.5-Flash and Claude Opus 4.8).

A.5. GDPval

For each task the deliverable produced by the harness is placed side by side with the human expert deliverable shipped with the benchmark, a panel of three judges of different provenance picks the better of the two, and the reported number is the win rate against the expert over 185 tasks. The panel combines an open-weight model served locally (Qwen3.6-35B-A3B (Qwen Team, 2026)) with two proprietary models from different vendors (Claude Sonnet 4.6 (Anthropic, 2026b) and Gemini-3.1

Pro (Google, 2026a)), each pair is judged in both presentation orders to remove position bias, and the verdict for a task is the majority vote of the three. Each judge therefore issues 204 comparisons per harness, and a win rate above 50% means the harness produces the preferred deliverable more often than the human expert it is compared against.

A.6. APEX-Agents

Each task places the agent in a sandboxed world with its own MCP tool surface, covering a filesystem, PDF reading, spreadsheets, mail, chat, calendar, documents and code execution, and spanning three professional domains. A task is graded by a per-task rubric judged by an LLM judge (Gemini-3.5-Flash), and a task counts as a success under pass@1 only when its rubric is satisfied on the single sampled rollout. We evaluate the full set of 480 tasks and always report over that full denominator, so a task whose rollout is missing because of an infrastructure failure counts as a failure rather than being excluded, which prevents a harness that crashes on hard worlds from looking better than one that attempts them.

A.7. EngDesign

We used the license-free subset of EngDesign, of which we take the 61 tasks that run without proprietary simulators. Each task states a design goal together with the physical constraints the design must satisfy, and each is graded by its own frozen simulation or testbench rather than by a judge model, so grading is deterministic and every point of variance we measure comes from the policy. Evolution runs on all 61 tasks with no in-distribution held-out split, since the suite is too small to spend tasks on one.

A.8. Frontier-Eng

Frontier-Eng collects real-world engineering optimization problems from 26 domains. Each task asks the agent to produce a design or a program that is scored by a frozen task-specific simulator or evaluator on a continuous objective, so as with EngDesign no judge model is involved and grading is deterministic. Because the objectives are not commensurable across tasks, the benchmark reports a Medal Score: for each task the three best feasible results of the frozen v1 snapshot are the gold, silver and bronze thresholds, a submission earns 1, 0.67 or 0.33 for reaching each, and the score is the mean credit over the 47 tasks of the v1 set, which we report as a percentage. We use Frontier-Eng only as an out-of-distribution test surface. Its EngDesign domain reuses tasks from our evolve set and is excluded, and tasks whose evaluation environment could not be built in our sandbox receive no credit in either arm, so 38 of the 47 tasks contribute credit and both arms are scored on exactly the same tasks.

B. Method Details

This section collects the round-level notation and procedural details omitted from Section 3 for readability. These definitions specify the same proposal- and selection-side regularizers described in the main text.

Algorithm 1 RRSI, proposal side.

Require: H_t, history $\mathcal{L}_t$, round t of T
Require: budgets $b_{\min}, b_{\max}$, window w, band δ
$\mathcal{F}_t \leftarrow \textsc{Analyze}(H_t, \mathcal{D}_{\text{evolve}})$ ▷ failure feedback
$b_t \leftarrow \left\lceil b_{\min} + (b_{\max} - b_{\min})\frac{1}{2}(1 + \cos\frac{\pi t}{T}) \right\rceil$ ▷ annealed L_0 budget, Eq. (4)
$\sigma_t \leftarrow \mathbb{1}[\hat{S}_t - \hat{S}_{t-w} \le \delta]$ ▷ stall flag
$\mathcal{T}_t \leftarrow \{\ell_i : (t_i, \ell_i, \ldots) \in \mathcal{L}_t\}$ ▷ tried
$\mathcal{U}_t \leftarrow \mathcal{K} \setminus \mathcal{T}_t$ ▷ untried components
$\mathcal{E}_t \leftarrow (\sigma_t, \mathcal{U}_t, m_{\text{draft}})$ ▷ Eq. (11)
$\mathcal{H}_t \sim P_{\text{reg}}(\cdot \mid H_t, \mathcal{F}_t, \mathcal{L}_t, b_t, \mathcal{E}_t)$
tag each $H' \in \mathcal{H}_t$ with (ℓ', h', d') ▷ component, hypothesis, diff
return $\{H' \in \mathcal{H}_t : \textsc{Critic}(H_t, H') = 1\}$ ▷ leakage screen, before evaluation

Algorithm 2 RRSI, selection side.

Require: screened $\mathcal{H}_t$, $(H_t, \hat{S}_t, \hat{C}_t)$, $S^\star$, δ, k
$\mathcal{A}_t \leftarrow \emptyset$
for $H' \in \mathcal{H}_t$ **in parallel do**
  $\hat{S}', \hat{C}' \leftarrow \textsc{Evaluate}(H', \mathcal{D}_{\text{evolve}}, k)$
  $\Delta S \leftarrow \hat{S}' - \hat{S}_t$; $\Delta C \leftarrow (\hat{C}' - \hat{C}_t)/\hat{C}_t$
  $c \leftarrow [\Delta C \le \beta_0 + \beta_1 \Delta S]$ if $\Delta S > \delta$, else $[w_s \Delta S - w_c \Delta C + w_n \nu(\ell') > 0]$ ▷ L_1 cost rule, Eq. (7)
  if $\hat{S}' \ge S^\star - \delta$ **and** c **then** ▷ floor Eq. (5)
    $\mathcal{A}_t \leftarrow \mathcal{A}_t \cup \{H'\}$
  end if
  append $(t, \ell', h', d', \Delta S, \Delta C, \mathbb{1}[H' \in \mathcal{A}_t])$ to $\mathcal{L}$
end for
$H_{t+1} \leftarrow \arg\max_{H' \in \mathcal{A}_t} \hat{S}'$, or H_t if $\mathcal{A}_t = \emptyset$
$S^\star \leftarrow \max(S^\star, \hat{S}_{t+1})$
return H_{t+1}, $\mathcal{B}_t = \{\ell \in \mathcal{T}_t : g_t(\ell) \le 0\}$ ▷ components to prune, Eq. (12)

B.1. Round-Level Formulation

Let $\Omega(H)$ denote the set of harnesses reachable from H by arbitrary source edits. RRSI leaves $\Omega(H)$ open and instead regularizes the transition through this space. A round takes the form

$$\mathcal{H}_t \sim P_{\text{reg}}\big(\cdot \mid H_t, \mathcal{F}_t, \mathcal{L}_t, b_t, \mathcal{E}_t\big) \subseteq \Omega(H_t), \qquad H_{t+1} = \underset{H' \in \mathcal{H}_t \cap \mathcal{A}_t}{\arg\max}\, \hat{S}(H'), \tag{8}$$

with $H_{t+1} = H_t$ if no candidate is admissible. Here $\mathcal{F}_t$ is the feedback from the current round, $\mathcal{L}_t$ is the edit history defined below, b_t is the annealed edit budget from Equation (4), $\mathcal{E}_t$ contains exploration directives, and $\mathcal{A}_t$ is the set of candidates allowed to replace the incumbent.

A run applies Algorithms 1 and 2 in turn for $t = 0, \ldots, T-1$, starting from H_0 with $S^\star = \hat{S}(H_0)$ and with the noise band δ estimated before evolution by repeatedly evaluating the unchanged base harness.

B.2. Proposal-Side Bookkeeping

Atomic edit representation. At round t, the proposer drafts a pool E_t of atomic edits to H_t, and a candidate is the subset of that pool it applies. Write this subset as $z_t \in \{0,1\}^{|E_t|}$, with $z_{t,j} = 1$ if edit j is included. The pool is redrawn each round from the open space $\Omega(H_t)$, so $|E_t|$ need not be fixed across rounds. The annealed budget in Equation (4) imposes $\|z_t\|_0 \le b_t$. Thus b_t limits the number of independent edits bundled into one candidate rather than the set of components that may eventually be modified.

Edit history and component-level summaries. Each candidate is tagged before evaluation with (ℓ', h', d'), where ℓ' names the harness component being modified, h' states the hypothesis tested by the edit, and d' is the source diff. The full history before round t is

$$\mathcal{L}_t = \{(t_i, \ell_i, h_i, d_i, \Delta S_i, \Delta C_i, a_i) : i \le n_t\}, \qquad a_i \in \{0,1\}, \tag{9}$$

where n_t is the number of candidates evaluated before round t, ΔS_i and ΔC_i are measured relative to the incumbent of round t_i as in Equation (6), and $a_i = 1$ if and only if the candidate was admitted.

Two summaries of this history are

$$\mathcal{T}_t = \{\ell_i : i \le n_t\}, \qquad g_t(\ell) = \max\{\Delta S_i : \ell_i = \ell,\ t - t_i \le n_{\text{prune}}\}, \qquad \max \emptyset = -\infty, \tag{10}$$

where $\mathcal{T}_t$ is the set of components already exercised and $g_t(\ell)$ is the best recent gain attributed to component ℓ.

Structured exploration state. Let $\mathcal{K}$ denote the editable harness components. The exploration directive is

$$\mathcal{E}_t = (\sigma_t, \mathcal{U}_t, m_{\text{draft}}), \qquad \sigma_t = \mathbb{1}[\hat{S}_t - \hat{S}_{t-w} \le \delta], \qquad \mathcal{U}_t = \mathcal{K} \setminus \mathcal{T}_t, \tag{11}$$

where σ_t indicates that progress over the previous w rounds is within the empirical noise band, $\mathcal{U}_t$ contains components not yet exercised, and m_{draft} reserves candidate slots for exploratory edits.

B.3. Selection-Side Bookkeeping

The leakage critic can be written as $\textsc{Critic} : \Omega(H_t) \to \{0, 1\}$ and is applied before full evaluation. Candidates that pass the critic are evaluated and filtered by the noise-adjusted floor in Equation (5) and the complexity-aware rule in Equation (7); the complete ordering of these checks is shown in Algorithm 2.

For structural pruning, a component belongs to the unproductive set

$$\mathcal{B}_t = \{\ell \in \mathcal{T}_t : g_t(\ell) \le 0\} \tag{12}$$

when it has been exercised but has produced no strictly improving candidate over the recent pruning window. These components are reported to the proposer as deletion targets in subsequent rounds.

C. Experiments

C.1. Hyperparameter Setting

RRSI introduces a small number of hyperparameters that control update sparsity, exploration, pruning, and the cost–performance trade-off. We select these parameters using only the evolve environment and operational considerations; held-out and OOD benchmarks are not used for tuning. The noise tolerance δ is calibrated from repeated evaluations of the unchanged base harness. The edit-budget parameters $(b_{\min}, b_{\max})$ determine how many independent changes can be bundled into one candidate, while w and m_{draft} control when and how strongly the search explores underused components. The pruning window n_{prune} determines how much recent evidence is required before a component is treated as unproductive. Finally, (β_0, β_1) encode the allowed trade-off between measured gain and additional inference cost. Table 5 lists the values used in each instance. Scores $\hat{S}$ are fractions in $[0, 1]$ and ΔC is the relative change in policy tokens per trial, so δ and β_1 are expressed in those units: on the coding instance δ corresponds to 3 passes out of $89 \times k = 178$ trials, on the agentic workspace instance to 60 criteria out of roughly 14,100 criterion verdicts, and on the engineering design instance to 5 passes out of $61 \times k = 244$ trials. Likewise β_1 corresponds to a 25% token allowance per additional pass (coding), per 100 additional criteria (agentic workspace) and a 10% allowance per additional pass (engineering design).

C.2. Hyperparameter Sensitivity

To test whether the reported gains depend on a narrow choice of these values, we additionally vary the most consequential regularization parameters around the default configuration while holding

Hyperparameter	Role	Coding	Agentic workspace	Engineering design
T	evolution rounds	20	20	40
k	trials per task per evaluation	2	2	4
δ	empirical noise tolerance	0.017	0.004	0.020
$b_{\min}$	final-round edit budget	1	1	1
$b_{\max}$	initial edit budget	4	3	4
w	stall-detection window	3	3	3
m_{draft}	reserved exploratory proposals	1	1	1
n_{prune}	pruning window	4	4	5
β_0	base cost allowance	0.10	0.10	0.15
β_1	gain-dependent cost allowance	44.5	35.4	24.4

Table 5 | Hyperparameters used by RRSI in each evolution setting. All choices are fixed without consulting held-out or OOD benchmarks.

Parameter group	Setting	Evolve score	Held-out/OOD score	Tokens/trial
Edit budget $(b_{\min}, b_{\max})$	smaller	TBD	TBD	TBD
	default	TBD	TBD	TBD
	larger	TBD	TBD	TBD
Stall window w	shorter	TBD	TBD	TBD
	default	TBD	TBD	TBD
	longer	TBD	TBD	TBD
Cost trade-off (β_0, β_1)	stricter	TBD	TBD	TBD
	default	TBD	TBD	TBD
	looser	TBD	TBD	TBD

Table 6 | Draft sensitivity analysis for the main RRSI hyperparameters. A compact version can report the OOD average used in Table 2; the final settings should be filled after running the corresponding sweeps.

the evolution budget fixed. We vary one parameter group at a time and report both evolve-set performance and transfer performance, since a setting that only increases the former may simply weaken regularization. Table 6 gives the reporting template for this analysis.

C.3. Detailed Ablation Studies

Variance in Final Performance Reporting. The empirical noise band used by RRSI during search and the uncertainty of a final benchmark score serve different purposes. The former controls whether a candidate's measured change is large enough to influence the evolution trajectory; the latter quantifies how much the reported performance of a fixed harness varies across repeated evaluations. We therefore report repeated-evaluation statistics for the final H_0 and RRSI harnesses in addition to the point estimates in the main text. Each pair is evaluated under matched decoding, tool, and judging conditions, and the same number of repetitions is used for the baseline and evolved harness.

For the evolution environments, we separately report the noise tolerance δ used by the search, because it should not be interpreted as an error bar on the final benchmark score. This distinction also makes clear whether the final transfer gain is large relative to ordinary evaluation variability rather than merely exceeding the search-time acceptance tolerance.

Domain	Benchmark	Role	H_0	RRSI	Δ	Repetitions
Coding	Terminal-Bench 2.1	Evolve	TBD ± TBD	TBD ± TBD	TBD	TBD
Coding	SWE-bench Verified	OOD	TBD ± TBD	TBD ± TBD	TBD	TBD
Agentic workspace	Harvey LAB	Evolve	TBD ± TBD	TBD ± TBD	TBD	TBD
Agentic workspace	Harvey LAB	ID held-out	TBD ± TBD	TBD ± TBD	TBD	TBD
Agentic workspace	JobBench	OOD	TBD ± TBD	TBD ± TBD	TBD	TBD
Agentic workspace	GDPval	OOD	TBD ± TBD	TBD ± TBD	TBD	TBD
Agentic workspace	APEX-Agents	OOD	TBD ± TBD	TBD ± TBD	TBD	TBD
Engineering design	EngDesign	Evolve	TBD ± TBD	TBD ± TBD	TBD	TBD
Engineering design	Frontier-Eng	OOD	TBD ± TBD	TBD ± TBD	TBD	TBD

Table 7 | Draft repeated-evaluation reporting for the final harnesses. Replace TBD entries with mean ± standard deviation (or the uncertainty measure used consistently in the paper) after repeated evaluations are completed.

Evolution benchmark	Base-harness repetitions	Calibrated δ	Calibration statistic
Terminal-Bench 2.1	TBD	TBD	TBD
Harvey LAB	TBD	TBD	TBD
EngDesign	TBD	TBD	TBD

Table 8 | Draft reporting of the search-time noise calibration. These values are distinct from the uncertainty estimates in Table 7.

C.4. Additional Seed-Evolve Validation

To test whether the transfer behavior depends on the particular benchmark used as the evolution seed, we add an independent evolution run starting from the same base harness but using a different seed benchmark. The new seed is kept disjoint from the benchmarks used to evaluate transfer, and the same candidate budget and regularization procedure are used. This experiment is intended to separate a property of RRSI from a fortunate choice of evolution benchmark: if the regularizers are doing their intended job, the resulting harness should again improve transfer rather than only the benchmark on which it was evolved.

A stronger version of this experiment uses two evolution seeds from the same broad domain and evaluates both resulting harnesses on the same untouched transfer benchmarks. This controls for domain and evaluation suite while changing only the benchmark that supplies adaptive evolution feedback. We will use this form when a second compatible seed benchmark is available.

Evolution seed	Evaluation benchmark	Role	H_0	RRSI
Existing seed	Existing evolve benchmark	Evolve	TBD	TBD
Existing seed	Common transfer benchmark A	OOD	TBD	TBD
Existing seed	Common transfer benchmark B	OOD	TBD	TBD
New seed (TBD)	New evolve benchmark	Evolve	TBD	TBD
New seed (TBD)	Common transfer benchmark A	OOD	TBD	TBD
New seed (TBD)	Common transfer benchmark B	OOD	TBD	TBD

Table 9 | Draft additional seed-evolve validation. The most informative comparison uses common held-out/OOD targets for the original and new evolution seeds, so that transfer can be compared without conflating it with a change in evaluation suite.