EVO^2: EVOLVING THE EVOLVER THROUGH WEIGHT-HARNESS CO-EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents are increasingly used to improve language model systems without human intervention through either weight updates or harness improvements. Existing approaches typically pursue these directions separately: some methods update model weights using self-generated experience, while others rewrite the harness that defines how an agent uses prompts, tools, context, and execution feedback. However, these approaches overlook the relationship between model weights and the agent harness. To connect these processes and ensure effective and smooth agent self-evolving, we propose EVO^2, a framework for weight-harness co-evolution. For harness updates, we design a unified dual-role pipeline that a single policy serves as both a Proposer, which writes harness edits, and a Reflector, which diagnoses their effects and guides revision. For weight updates, harness modifications and their execution outcomes guide the training of both the Proposer and the Reflector. We further develop EVOTIDE, a training pipeline that recovers usable trajectories through edit extraction and prefix continuation, then combines SFT with interleaved GRPO across the two roles. Extensive experiments across three coding benchmarks demonstrate that EVO^2 significantly outperforms existing self-evolution methods. It achieves 82.1% on Polyglot and 48.3% on SWE-bench Verified, compared with 62.6% and 41.0% for the strongest baseline, respectively. We also show that the evolved harnesses transfer to different tasks without further evolution and rank first in cross-benchmark evaluations. Our dual-role harness improvement framework is also able to extend to self-evolution with proprietary models, improving a GPT-5-mini agent's performance from 24.4% to 48.3%. In addition, EVO^2 generalizes to other tasks smoothly. We extend self-evolution to memory tasks, achieving the best performance on the LoCoMo and LongMemEval benchmarks. These results demonstrate the applicability of the framework EVO^2 across domains and its broad transferability across models and tasks.

1 INTRODUCTION

Self-evolving agents offer a way to improve language model systems through their own experience and without human intervention. Existing approaches typically follow two directions. The first updates *model weights*: a model generates solutions, selects useful outputs using task feedback, and trains on them to improve its ability to solve similar problems (Zelikman et al., 2022; Singh et al., 2024). The second modifies the *agent harness*: a model inspects and rewrites the program that defines how an agent uses prompts, tools, context, and execution feedback (Robeyns et al., 2025; Zhang et al., 2026). By evaluating these edits and retaining useful changes, an agent improves its task performance through modifications to its own code. These approaches accumulate experience in model parameters and executable programs, respectively, but typically pursue the two forms of improvement separately.

Treating these directions separately overlooks the relationship between model weights and the agent harness. The model's capabilities shape the modifications it proposes and how it interprets execution feedback. In turn, each evaluated modification reveals which changes are useful, why an edit fails, and how different improvements interact. This experience can guide the training of the model responsible for evolution. In particular, proposing edits and diagnosing their effects are skills that can themselves be learned from earlier evolution attempts, connecting harness improvement to weight updates.

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.

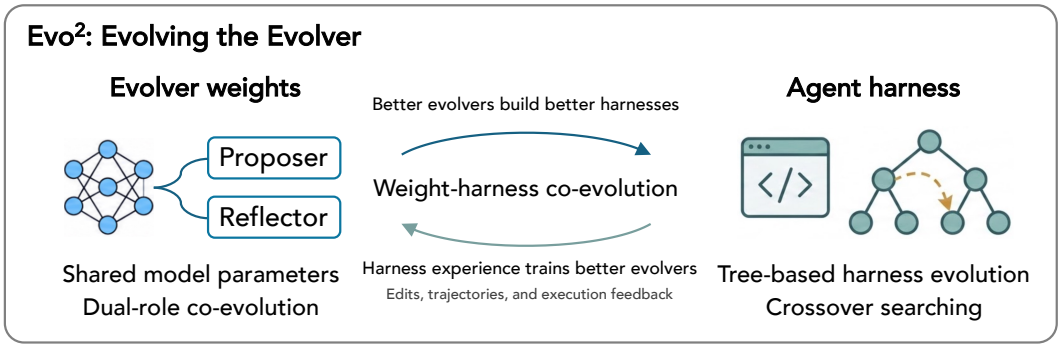


Figure 1: **Weight-harness co-evolution in EVO².** A shared Proposer–Reflector policy evolves the agent harness. Harness edits and execution outcomes provide training experience for the policy, whose updated weights guide subsequent search. Crossover searching transfers useful strategies between branches.

We introduce EVO², a framework for *weight-harness co-evolution* in self-evolving agents (Figure 1). We train the model to become a better *evolver*: to propose useful harness modifications, interpret their outcomes, and revise them accordingly. A better evolver builds better harnesses, whose modifications and execution outcomes provide further experience for training the evolver. Harness evolution, therefore, supplies training signals for weight updates, while the updated model guides subsequent harness search.

For harness updates, EVO² uses a unified dual-role pipeline in which a single policy serves as both a *Proposer* and a *Reflector*. The Proposer writes harness edits, while the Reflector examines their execution outcomes and provides feedback for further revision. Both roles share parameters and learn from the effects of the modifications they produce. We organize the search as a tree of harness variants and introduce *crossover searching* to combine useful strategies discovered on different branches. The Reflector identifies a strategy that addresses a weakness in the current harness, and the Proposer adapts it to that harness, allowing each branch to build on discoveries made elsewhere in the tree.

For weight updates, we develop EVOTIDE (Tree-guided Interleaved Dual-role Evolution) for weight updates, to train the shared policy on experience collected during harness evolution. We collect trajectories from both the Proposer and the Reflector, using edit extraction and prefix continuation to recover usable trajectories from incomplete or partially invalid rollouts. These trajectories support an SFT warm start, followed by interleaved GRPO across the two roles. A maintained parent pool exposes the policy to harnesses at different depths and performance levels, so it learns both to improve strong candidates and to repair weaker ones. The updated policy then generates the proposals and feedback used in subsequent evolution rounds.

Extensive experiments across three coding benchmarks demonstrate that EVO² outperforms existing self-evolution methods. It achieves 82.1% on Polyglot and 48.3% on SWE-bench Verified, compared with 62.6% and 41.0% for the strongest respective baselines. The evolved harnesses also transfer across benchmarks without further evolution. We further apply the dual-role harness improvement framework to proprietary models with fixed weights and extend self-evolution to memory tasks, achieving the best performance on LoCoMo and LongMemEval. These experiments demonstrate both the applicability of the framework across domains and the transferability of its evolved harnesses.

Our contributions are:

- **Weight-harness co-evolution:** We introduce EVO², which connects harness evolution with the training of a unified Proposer–Reflector policy through harness modifications and their execution outcomes.
- **Crossover searching:** We combine reflection and directed editing to adapt useful strategies across branches of the harness tree.
- **A training pipeline for the evolver:** We develop EVOTIDE, which combines trajectory recovery, SFT, and interleaved GRPO with a parent pool spanning different depths and performance levels.

- **A reusable framework and evolved harnesses:** We demonstrate self-evolution on coding and memory tasks, harness transfer across benchmarks, and harness improvement with fixed proprietary models.

2 Related Work

Self-improving language models. Self-improvement through training forms an iterative loop: a model generates experience, learns from feedback, and uses its updated weights to collect the next round of training data (Gao et al., 2025; Fang et al., 2025). STaR and ReST$^{\text{EM}}$ train on successful model outputs (Zelikman et al., 2022; Singh et al., 2024), while Absolute Zero and R-Zero develop tasks and solvers through self-play (Zhao et al., 2026; Huang et al., 2026). Other methods improve training signals through trajectory selection, semantic voting, or learned evaluators (Zhong et al., 2026; Jiang et al., 2026; Wang et al., 2024). EvolveR combines reusable experience with policy reinforcement (Wu et al., 2025). ReVeal, Agent0-VL, and Self-Distillation Zero train generation together with verification or revision (Jin et al., 2026; Liu et al., 2025; He et al., 2026); EvoLM and CoNL co-train generation and feedback mechanisms within shared policies (Li et al., 2026; Sui & Hooi, 2026). Evo^2 applies shared-role training to harness modification: EvoTIDE uses supervised fine-tuning and interleaved GRPO to train a unified Proposer–Reflector model on evaluated edits and their outcomes.

Self-evolving agent harnesses. Agents can also accumulate improvements in the harness that manages their prompts, tools, and control flow. They propose modifications, evaluate the resulting agent, and use the outcomes to guide further changes. SICA edits its own implementation (Robeyns et al., 2025), DGM maintains an archive for branching self-modification (Zhang et al., 2026), and HGM selects promising parents using estimated descendant performance (Wang et al., 2026). Meta-Harness searches over code using prior candidates and execution traces (Lee et al., 2026). GEPA combines reflection with prompt evolution (Agrawal et al., 2026), while GEA shares experience across evolving agents (Weng et al., 2026). The Red Queen Gödel Machine extends evolution to changing evaluation utilities (Iacob et al., 2026). Prior work also demonstrates harness transfer across tasks and models (Wang et al., 2026; Lee et al., 2026; Weng et al., 2026). Evo^2 makes harness search a source of self-improvement for the evolver itself: crossover searching lets the Reflector select a donor strategy and the Proposer adapt it to a recipient harness, while the resulting experience trains both roles to guide subsequent search. We evaluate the framework's applicability across coding and memory domains as well as the transfer of its evolved harnesses across benchmarks and executors.

3 Method

3.1 Overview

Evo^2 integrates harness evolution with weight evolution (Figure 2). A harness, h, is the executable program that manages an agent's prompts, tools, context, and control flow. A language model policy, π_θ, serves as the evolver, acting as both Proposer and Reflector through different role prompts. We train its parameters, θ, on the outcomes of harness modifications. The task executor, denoted M_{exec}, runs inside each candidate harness and remains fixed during co-evolution, so candidate performance measures the effect of changing the harness.

Harness evolution proceeds through iterative tree search. Each node stores a harness and its task evaluation records; an edge records the edits made to its parent. At each iteration, we sample a parent, use the Proposer to modify it, and evaluate the resulting candidate. The Reflector then examines the execution feedback and recommends a revision, integration of a strategy from another branch, or retention of the candidate. The Proposer implements any recommended change, which is evaluated in turn. The resulting harnesses extend the tree, while the role trajectories and task outcomes provide data for weight evolution. EvoTIDE uses these data for supervised fine-tuning followed by interleaved reinforcement learning. Updated weights are used in subsequent rounds of search and data collection.

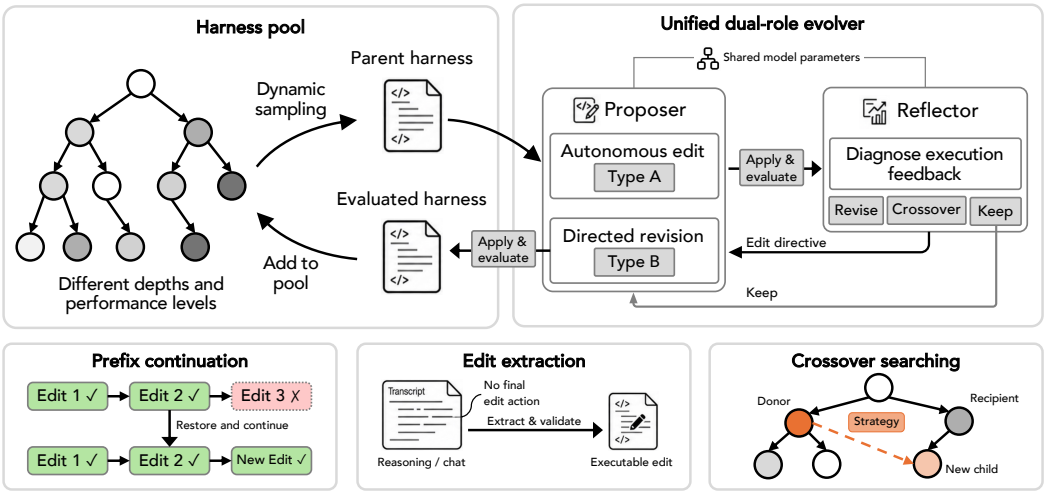


Figure 2: **Harness evolution and trajectory collection.** Parents sampled from the harness pool receive autonomous Type A edits from the Proposer and are evaluated with a fixed task executor. The Reflector uses execution feedback to direct revision or crossover, implemented through Type B edits, or to keep the candidate without further editing. Both roles share model parameters, and evaluated candidates extend the pool. The lower panels illustrate prefix continuation, edit extraction, and crossover searching.

3.2 Harness evolution through dual-role self-modification

Proposer. The Proposer operates in two modes. In *Type A*, it receives a parent harness h_p and its measured task performance, chooses a modification, and produces edits e_0 to obtain an initial candidate h_0. In *Type B*, it receives a harness and an explicit edit directive d from the Reflector. It implements that directive against the current source through edits e_1, producing a revised candidate h_1. Type A develops the ability to identify improvements independently; Type B develops the ability to turn feedback into working code. Both modes can generate a sequence of file edits. startup before being evaluated on tasks.

Reflector. The Reflector receives the candidate source, the edits relative to its parent, the change in task performance, and execution traces from relevant successes and failures. It identifies the strategy attempted by the Proposer, diagnoses remaining problems, and chooses one of three actions: REVISE provides a directive to improve the candidate; CROSSOVER provides a directive to incorporate a strategy from another branch; and KEEP retains the candidate without another edit. A revision directive describes a concrete change and the execution evidence that motivates it. Its quality is measured by the performance of the harness produced when the Proposer follows it. This also allows the Reflector to learn when an additional modification is unlikely to help.

Let z_p contain the parent's evaluation records and z_0 contain the candidate's edit diff and execution feedback. With available donor strategies $\mathcal{S}$, the shared policy implements the cycle

$$\begin{aligned} e_0 &\sim \pi_\theta(\cdot \mid P_A, h_p, z_p), & h_0 &= \mathrm{Apply}(h_p, e_0), \\ (m, d, j) &\sim \pi_\theta(\cdot \mid R, h_p, h_0, z_0, \mathcal{S}), & & \\ e_1 &\sim \pi_\theta(\cdot \mid P_B, h_0, d), & h_1 &= \mathrm{Apply}(h_0, e_1). \end{aligned} \tag{1}$$

Here P_A, P_B, and R specify role prompts; all calls use the same model parameters θ. We write the Reflector's decision compactly as its action m, directive d, and donor identifier j, which is used only for crossover. The last line is executed for revision or crossover; for KEEP, $h_1 = h_0$ and no further edit is sampled. During training, counterpart calls use the frozen snapshot described in Section 3.3.3.

Crossover searching. Different branches may discover complementary improvements. To make these discoveries available to the Reflector, we maintain a library $\mathcal{S} = \{s_j\}$ of strategies extracted from evaluated harnesses. Each entry describes a mechanism and its observed effect, together with

whether it is already present in the current candidate. Given this context, the Reflector can select a donor strategy that addresses a problem revealed by the candidate's execution. Its directive d specifies how to integrate the selected s_j into h_0. The Type B Proposer then adapts the mechanism to the candidate's source and combines it with the existing behavior. We evaluate the combined harness to determine whether the integration helps. A crossover child retains the recipient as its tree parent and records the donor strategy, so it can receive further revision or crossover.

3.3 WEIGHT EVOLUTION THROUGH EVOTIDE

EVOTIDE turns the trajectories generated during harness evolution into training data for the shared policy. We first collect an initial dataset and perform supervised fine-tuning (SFT). We then continue collecting experience from the evolving tree while training both roles with group relative policy optimization (GRPO) (Shao et al., 2024).

3.3.1 DATA COLLECTION

For each evolution attempt, we record the role prompts, model responses, applied edits, parent and child harnesses, and task evaluation results. One propose-evaluate-reflect-revise cycle provides an autonomous Proposer trajectory, a Reflector trajectory, and, when a revision is requested, a directed Proposer trajectory. The same evaluated harness can also serve as a parent in a later round or contribute a strategy to the crossover library.

Recovering Proposer trajectories. An otherwise useful rollout may fail to produce an executable edit because of how the response is expressed. We use two recovery procedures during data collection. *Edit extraction* handles rollouts that end without an explicit edit action: we search the recorded reasoning and chat messages for concrete file edits and convert them into the required action format. *Prefix continuation* handles a sequence of edits in which a later edit is invalid, for example because of a malformed tool call or an incorrect edit format. We truncate the rollout after the last successfully applied edit, restore the corresponding intermediate harness, and resume generation from that prefix. This preserves the valid earlier edits while allowing the model to regenerate the failed suffix. Recovered proposals pass the same application, startup, and task evaluation checks as other candidates before entering the training data.

Collecting Reflector trajectories. Evaluated proposals become reflection cases, including edits that improve performance, leave it unchanged, or cause regressions. For each case, we generate feedback and evaluate the candidate obtained by implementing it; a KEEP response retains the initial candidate. These outcomes provide supervision for the Reflector's diagnosis and choice of action. Connecting strategies from other branches adds crossover cases, so the dataset includes both local revisions and the integration of previously discovered mechanisms.

3.3.2 REWARD DESIGN

Both roles are rewarded by the change in task performance caused by their decisions. Let $\widehat{V}(h) \in [0, 1]$ be the mean externally evaluated task score of harness h; for tasks with a binary correctness signal, this is its success rate. For a valid Proposer edit from h to h', and a valid Reflector intervention from h_0 to h_1, the execution rewards are

$$r_P^{\text{exec}} = \widehat{V}(h') - \widehat{V}(h), \qquad r_R^{\text{exec}} = \widehat{V}(h_1) - \widehat{V}(h_0). \tag{2}$$

For Type A, the Proposer is compared with its sampled parent; for Type B, it is compared with the candidate it was instructed to revise. A KEEP action has $h_1 = h_0$ and receives zero execution reward. We evaluate each comparison on the same tasks under matched execution conditions, averaging repeated runs when available.

Malformed outputs and edits that cannot be applied or started receive a negative reward. No-op edits, unusable harnesses, and invalid crossover directives are also penalized. Type B receives a small additional bonus or penalty according to whether the edit touches a file named in the directive. For Reflector interventions, positive gains are credited only when the traces contain evidence that the added mechanism ran; otherwise the gain is set to zero.

3.3.3 TRAINING PROCESS

Supervised fine-tuning. We initialize the shared policy with a mixture of successful autonomous edits, directed edits, and Reflector responses selected using their downstream evaluation results. The model is trained to predict the recorded response from its role prompt and harness context. We balance the training mixture across roles and modes to establish both the editing and reflection behavior needed for online training.

Tree maintenance and parent sampling. During online training, evaluated candidates extend the harness tree and refresh a bounded parent pool. We retain nodes at multiple depths so the model encounters both early modifications and harnesses with several accumulated changes. Within each depth, we reserve part of the pool for the strongest candidates and sample the remainder from other executable candidates. This retains weaker harnesses that need repair as well as strong harnesses that can be improved further. Let $\mathcal{P}_r$ be the retained pool at round r, and let $\mathcal{N}_r = \mathcal{P}_r \setminus \{h_{\text{root}}\}$ contain its non-root harnesses. The root has a permanent place and an allocated sampling share $\rho \in (0, 1)$. With weights $w(h) = \epsilon_p + |\widehat{V}(h) - \frac{1}{2}|$, where $\epsilon_p > 0$, parent samples follow

$$p_r(h) = \begin{cases} \rho, & h = h_{\text{root}}, \\ (1-\rho)\dfrac{w(h)}{\sum_{u \in \mathcal{N}_r} w(u)}, & h \in \mathcal{N}_r. \end{cases} \tag{3}$$

If $\mathcal{N}_r$ is empty, we sample only the root. This distribution gives both high and low-scoring parents more exposure while keeping every retained parent available. Updating the pool across rounds lets the training distribution develop with the harnesses produced by the model.

Interleaved role training. We optimize a single policy for both roles. At the beginning of training round r, we set $\bar{\theta}_r = \theta_r$ and freeze $\pi_{\bar{\theta}_r}$ to provide the counterpart behavior needed to evaluate training samples. For Proposer training, this copy provides Reflector directives for Type B prompts. For Reflector training, it acts as the Proposer that implements sampled directives. Each optimization step uses a Proposer group and a Reflector group, with advantages computed separately for each prompt. Both role updates change the live parameters θ, while $\bar{\theta}_r$ stays fixed. After the round yields θ_{r+1}, we refresh the snapshot by setting $\bar{\theta}_{r+1} = \theta_{r+1}$ and collect new cases. An improved Proposer therefore generates new modifications for the Reflector to learn from, while an improved Reflector generates new directives for the Proposer. Both skills improve through successive rounds of this interaction.

Balancing modes within each role. We interleave Type A and Type B Proposer prompts so that training covers both autonomous improvement and implementation of feedback. For the Reflector, we sample half of the inputs with an empty strategy set and half with crossover strategies. We also construct paired SFT prompts: when a target response is valid in either context, the same response is used with and without the strategy set. This teaches the model to retain a useful diagnosis even when donor information is unavailable. Targets that depend on a specific crossover donor are used only with the corresponding strategy context. The mixture includes revision, crossover, and retention decisions, and is refreshed as new trajectories become available.

Policy optimization. For a role prompt q, we sample a group of G responses $y_1, \ldots, y_G$ from the current policy and compute rewards $r_1, \ldots, r_G$. We use the group mean baseline $A_i = r_i - \frac{1}{G}\sum_{j=1}^{G} r_j$ and normalize the token losses by a fixed maximum response length $L_{\max}$. We average the losses from the two role groups and take one update per sampled batch. Truncated responses are masked from the loss. All responses within a group share the same role and input, so their advantages compare alternative edits or reflections for the same harness context.

The trained evolver can run the same search procedure on a new initial harness given an evaluator for its tasks. An evolved harness can also be deployed directly on new tasks with compatible model and tool interfaces. These two forms correspond to the outputs of co-evolution: weights that encode how to improve a harness, and harnesses that incorporate the improvements found during search.

Table 1: Self-evolution on coding benchmarks. Each column reports the held-out performance of the harness evolved on that benchmark, using Qwen3.8-27B. All scores are percentages; higher is better.

Method	Polyglot Pass@1 ↑	SWE-bench Verified Resolved ↑	LiveCodeBench Pass@1 ↑
SICA	60.2	20.5	39.0
DGM	60.2	30.0	54.0
HGM	61.8	41.0	56.0
GEA	62.6	25.0	49.0
EVO^2 (ours)	**82.1**	**48.3**	**58.0**

Table 2: Transfer across coding benchmarks, using Qwen3.8-27B. The upper header identifies the benchmark used to evolve and select the best harness; the lower header identifies its evaluation benchmark. Each method contributes one harness per source, evaluated on both other benchmarks without further evolution. Scores use the target benchmark's metric from Table 1.

	From Polyglot		From SWE-BenchVerif.		From LCB	
Method	SWE	LCB	Polyglot	LCB	Polyglot	SWE
SICA	19.0	33.0	52.0	49.0	43.9	29.0
DGM	20.0	**45.0**	69.9	45.0	63.4	19.0
HGM	20.0	41.0	53.7	50.0	63.4	16.0
GEA	16.0	**45.0**	48.8	42.0	61.0	25.0
EVO^2 (ours)	**21.0**	44.0	**72.4**	**53.0**	**74.0**	**43.0**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Models. Our main experiments use Qwen3.8-27B, served in FP8 with greedy decoding, as the fixed task executor. We use the same model as the evolver: the untrained base model for the baselines and the model fine-tuned by EVOTIDE for EVO^2, where a single policy serves as both Proposer and Reflector. For memory tasks, we additionally use a Qwen3-Embedding-0.6B retriever. In the proprietary-model experiments, GPT-5-mini and Claude Haiku 4.5 each serve as both executor and evolver, with their weights fixed throughout evolution.

Benchmarks and metrics. We evaluate coding harness evolution on Polyglot, SWE-bench Verified (SWE), and LiveCodeBench (LCB). We report held-out pass@1 on Polyglot tasks over three evaluation runs, the resolved rate on SWE-bench Verified instances, and pass@1 on LCB problems. Proprietary models are evaluated on a 60-task SWE-bench Verified subset. For memory tasks, we evolve harnesses on LoCoMo and report mean F1 on three held-out conversations. We also evaluate the same harnesses on LongMemEval-S using accuracy over 500 instances, with GPT-based judging and three evaluation runs.

Baselines. We compare against four representative self-evolving agent frameworks: **SICA** (Robeyns et al., 2025), which edits its own implementation along a single lineage; **DGM** (Zhang et al., 2026), which maintains an open-ended archive and branches self-modifications from stored agents; **HGM** (Wang et al., 2026), which selects parents based on the estimated performance of their descendants; and **GEA** (Weng et al., 2026), which enables concurrently evolving agents to share experience. All baselines use the untrained model as the evolver and follow their own selection rules.

4.2 SELF-EVOLUTION ON CODING BENCHMARKS

EVO^2 evolves the strongest harness on every benchmark. Table 1 reports the held-out performance of the best harness evolved by each method on each benchmark. On Polyglot, EVO^2 achieves 82.1% pass@1, compared with 62.6% for the strongest baseline, GEA. On SWE-bench Verified, it resolves 48.3% of held-out instances, compared with 41.0% for HGM. On LiveCodeBench, EVO^2 reaches 58.0% pass@1, while the four baselines range from 39.0% to 56.0%. These results demonstrate consistent improvements across all three coding benchmarks.

4.3 HARNESS TRANSFER ACROSS BENCHMARKS

For each method and source benchmark, we take the harness reported in Table 1 and evaluate it on the other two benchmarks. Table 2 reports all six transfer directions.

EVO^2 achieves the best transfer performance in five of six directions. The harness evolved on SWE-bench Verified achieves 72.4 on Polyglot and 53.0 on LCB, while the harness evolved on LCB achieves 74.0 on Polyglot and 43.0 on SWE-bench Verified. In each case, EVO^2 outperforms every baseline, with margins ranging from 2.5 to 18 points. These results show that the improvements discovered during evolution remain useful beyond the tasks used to select the harness.

4.4 SELF-EVOLUTION WITH PROPRIETARY MODELS

Table 3 reports self-evolution results on the 60-task SWE-bench Verified subset with GPT-5-mini and Claude Haiku 4.5. In each column, the named model performs both task execution and harness evolution. For EVO^2, it serves as executor, Proposer, and Reflector with fixed weights; the dual-role reflection and crossover search therefore constitute the differences from the baselines. We conduct three independent searches per method and model on a small search subset and report the mean over seeds. Each baseline evaluates its top-ranked harness, while EVO^2 reports the better result of its two top-ranked harnesses.

Table 3: Self-evolution with proprietary models on a 60-task SWE-bench Verified subset (resolved, %). In EVO^2, each model serves as executor, Proposer, and Reflector with fixed weights.

	Executor and evolution model	
Method	GPT-5-mini	Claude Haiku 4.5
SICA	24.4	42.2
DGM	23.9	43.3
HGM	24.3	45.0
GEA	19.4	44.4
EVO^2 (ours)	**48.3**	**47.8**

Dual-role search improves proprietary agents with fixed weights. With GPT-5-mini, EVO^2 achieves a 48.3% resolved rate, compared with 19.4–24.4% for the baselines, nearly doubling the strongest baseline's performance. With Claude Haiku 4.5, whose unevolved harness already performs strongly, EVO^2 reaches 47.8%, compared with 42.2–45.0% for the baselines. These results demonstrate that harness search provides improvements on its own. When the evolver can be trained, weight evolution provides additional gains, as shown in Section 4.6.

4.5 SELF-EVOLUTION ON MEMORY TASKS

To evaluate the framework in another domain, we instantiate it with a memory harness and perform self-evolution on LoCoMo. We retain the Proposer–Reflector interaction, crossover search, and training procedure, while adapting the editable harness and task evaluator to memory tasks. Each search evaluates 1,500 questions drawn from seven training conversations. Table 4 reports the results.

Table 4: Self-evolution on memory tasks, using Qwen3.8-27B. LoCoMo: held-out F1 of the harness evolved and selected on LoCoMo. LongMemEval: accuracy of the same harness on LongMemEval-S without further evolution or selection. Both are percentages averaged over independent searches.

Method	LoCoMo F1 ↑	LongMemEval Accuracy ↑
SICA	54.6	59.7
DGM	55.8	60.3
HGM	56.2	64.4
GEA	53.2	48.9
EVO^2 (ours)	**60.4**	**71.5**

EVO^2 outperforms the strongest baseline on LoCoMo and transfers better to LongMemEval. On LoCoMo, EVO^2 achieves 60.4 F1, compared with 56.2 for HGM and 53.2–55.8 for SICA, DGM, and GEA. On LongMemEval, which is unseen during both evolution and selection, every EVO^2 search achieves a score between 67.9 and 73.9, exceeding the HGM mean of 64.4. GEA performs below the other baselines.

4.6 IN-DEPTH ANALYSIS OF EVO^2

Weight evolution improves the evolver on every benchmark. Table 5 isolates the contribution of EVOTIDE. Each comparison fixes the search procedure, the executor, and the parent harnesses, varying only whether the evolver's weights are trained. The trained evolver produces stronger champions on Polyglot (+16.3 points) and LoCoMo (+9.3), while its LoCoMo champion transfers better to LongMemEval (+10.6). Crossover search provides a further 4.1-point improvement on Polyglot in Table 1. The trained evolver produces better children on average.

Table 5: Effect of weight evolution. Each row compares the untrained evolver (the base Qwen3.8-27B acting as Proposer and Reflector) with the evolver trained by EvoTIDE.

Benchmark	Untrained	Trained	Δ (P)
Polyglot	61.8	**78.0**	+16.3 (.001)
SWE-bench Verified	46.0	**51.0**	+5.0 (.086)
LoCoMo	47.2	**56.5**	+9.3
LongMemEval	57.1	**67.7**	+10.6

Performance gains do not require additional inference compute. Figure 3 compares held-out LiveCodeBench accuracy with the inference cost of each evolved harness. The Evo^2 harness uses 3.6K completion tokens and 10.2 LLM calls per task, close to the unevolved harness at 2.9K tokens and 8.7 calls. HGM and DGM achieve lower accuracy despite using 6.6K–6.9K tokens and 26–37 calls per task. The same pattern holds on Polyglot, where our champion uses fewer calls and tokens than the unevolved harness: 36 versus 43 calls and 3.6K versus 4.0K tokens per task. These results show that harness evolution improves how effectively the agent uses a fixed inference budget.

Development-set scores are computed on a small task set that is repeatedly optimized during search, making the highest ranks particularly susceptible to selection noise. Retaining the two top-ranked harnesses, as Evo^2 does, recovers most of this loss at the cost of one additional held-out evaluation. The Reflector's diagnosis, grounded in execution feedback, is designed to reduce the same noise during search.

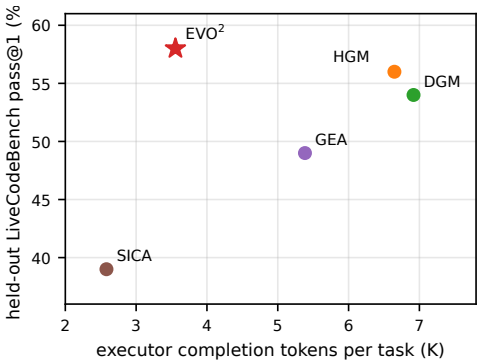


Figure 3: **Held-out accuracy versus inference cost.** Held-out LiveCodeBench pass@1 of the harnesses in Table 1, plotted against mean executor completion tokens per held-out task. Evo^2 achieves the highest accuracy, while HGM and DGM use about $1.9\times$ as many tokens and $2.6\times$ and $3.6\times$ as many LLM calls per task, respectively.

5 Conclusion

We presented Evo^2, a framework for weight-harness co-evolution in self-evolving agents. A single policy serves as both Proposer and Reflector, learning from harness modifications and their execution outcomes to improve its ability to propose edits and guide revision. Crossover searching combines useful strategies across branches, while EvoTIDE trains the shared policy through trajectory recovery, SFT, and interleaved GRPO. Experiments on coding and memory tasks demonstrate improved task performance and transfer of evolved harnesses across benchmarks. The dual-role search also supports harness improvement with fixed proprietary models. Our analyses show that weight evolution improves search reliability and that crossover searching reduces the evaluation budget needed to reach strong performance. Evolving the evolver allows experience to accumulate in both the executable harness and the policy that guides future evolution.

AI use statement

In this work, we used generative AI tools for generating synthetic data sets. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, clean and reformat dataset, support qualitative and thematic data analysis, interpret results, and assisting with translation are not applicable to this work. We have reviewed all AI-assisted work. LLM-generated data was verified and tested for correctness by 2 authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-distillation zero: Self-revision turns binary rewards into dense supervision. *arXiv preprint arXiv:2604.12002*, 2026.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026.

Alex Iacob, Andrej Jovanović, William F Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Ambroise Odonnat, Zeyu Cao, et al. The Red Queen Gödel Machine: Co-evolving agents and their evaluators. *arXiv preprint arXiv:2606.26294*, 2026.

Chunyang Jiang, Yonggang Zhang, Yiyang Cai, Chi-Min Chan, Yulong Liu, Mingming Chen, Wei Xue, and Yike Guo. Semantic voting: A self-evaluation-free approach for efficient llm self-improvement on unverifiable open-ended tasks. In *International Conference on Learning Representations*, volume 2026, pp. 36536–36556, 2026.

Yiyang Jin, Kunzhao Xu, Hang Li, Xueting Han, Yanmin Zhou, Cheng Li, and Jing Bai. Reveal: Self-evolving code agents via reliable self-verification. In *International Conference on Learning Representations*, volume 2026, pp. 149800–149824, 2026.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026.

Shuyue Stella Li, Rui Xin, Teng Xiao, Yike Wang, Rulin Shao, Zoey Hao, Melanie Sclar, Sewoong Oh, Faeze Brahman, Pang Wei Koh, et al. Evolm: Self-evolving language models through co-evolved discriminative rubrics. *arXiv preprint arXiv:2605.03871*, 2026.

Jiaqi Liu, Kaiwen Xiong, Peng Xia, Yiyang Zhou, Haonian Ji, Lu Feng, Siwei Han, Mingyu Ding, and Huaxiu Yao. Agent0-vl: Exploring self-evolving agent for tool-integrated vision-language reasoning. *arXiv preprint arXiv:2511.19900*, 2025.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2312.06585.

Yuan Sui and Bryan Hooi. Conversation for non-verifiable learning: Self-evolving llms through meta-evaluation. *arXiv preprint arXiv:2601.21464*, 2026.

Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. Self-taught evaluators. *arXiv preprint arXiv:2408.02666*, 2024.

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel Machine: Human-level coding agent development by an approximation of the optimal self-improving machine. In *International Conference on Learning Representations*, volume 2026, pp. 80356–80386, 2026.

Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing. *arXiv preprint arXiv:2602.04837*, 2026.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

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

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, Leszek Rutkowski, and Dacheng Tao. Better, faster: Harnessing self-improvement in large reasoning models. In *Forty-third International Conference on Machine Learning*, 2026.