EVIGATE: EVIDENCE-GATED INHERITANCE FOR SELF-EVOLVING PROGRAM-REPAIR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents can generate patches, execute tests, and reuse repair trajectories, making program repair a natural setting for studying agents that adapt from their own interaction histories. Yet passing the evidence available on one task does not establish a reliable improvement: additional rollouts help only when their outcomes can be selected correctly, while local lessons may overfit visible tests and cause regressions when inherited through memory or parameters. We introduce **EviGate**, an evidence-gated inheritance framework that treats self-evolution as a sequence of separately authorized state transitions. EviGate combines execution-grounded candidate selection, staged experience consolidation, and parameter adaptation whose proposed children are retained only after promotion, regression, and safety checks, with append-only lineage connecting every decision. On 100 seeds and 17 fixed EVOCODEGYM tasks, uninformed three-candidate sampling adds only $+0.39/75$ private cases, whereas executable selection gains $+11.05$ (seed$\times$task 95% CI $[7.22, 15.22]$). Memory, parameter, fresh-task, and repository-level evaluations further expose negative transfer and weak proposal coverage, establishing EviGate as an auditable inheritance protocol and empirical boundary map rather than a demonstration of repeated capability growth.

1 INTRODUCTION

Large language models (LLMs) are increasingly embedded in software-engineering agents that inspect repositories, edit files, invoke tools, and execute tests. SWE-bench shifted evaluation from isolated code completion to resolving real repository issues (Jimenez et al., 2024); systems including SWE-agent, RepairAgent, and AutoCodeRover subsequently showed the value of agent–computer interfaces, autonomous tool use, structured code search, and environment interaction (Yang et al., 2024; Bouzenia et al., 2025; Zhang et al., 2024). At the same time, the competitive Agentless baseline cautions that gains should not be attributed to agentic complexity without controlled comparison (Xia et al., 2025). Conversational repair and generated-test filtering further demonstrate that compiler and test feedback can guide revision and improve patch precision (Xia & Zhang, 2024; Mündler et al., 2024). These developments make program repair a particularly attractive setting for studying agent self-evolution: repair attempts produce executable, replayable outcomes that can inform not only the current decision, but also the agent's future memory and parameters. In principle, an agent can generate a patch, evaluate it, extract what was learned, update itself, and use the resulting version to begin the next repair cycle.

Closing this loop, however, requires more than learning from every successful trajectory. A candidate can pass visible tests because of stochastic luck, incomplete coverage, or behavior tailored to the observed cases, and a plausible lesson abstracted from that candidate may fail on related tasks. The risk grows as an update becomes more persistent: selecting a patch affects one repair, storing a rule affects future contexts, and training on that rule changes the model across tasks. Experiential agents show that textual reflection, accumulated insights, and executable skill libraries can improve subsequent decisions without modifying model weights (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024); later work learns reusable instructions, structured memories, or hierarchical workflows from collections of trajectories (Gupta et al., 2024; Xu et al., 2025; Fan et al., 2025). Context-evolution methods extend this idea by continually curating knowledge into editable playbooks (Zhang et al., 2026b). Yet a trajectory's usefulness on the task that produced it does not by itself establish that the

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.

extracted experience is safe to inherit. Reliable self-evolution must therefore distinguish four decisions that are often treated as one: what behaviors to explore, which behavior to select now, which experience to retain, and which retained experience is sufficiently supported to update the agent.

Existing self-improvement approaches instantiate parts of this process but often grant comparable authority to different evidence. Rollout feedback can evolve prompts, generate adaptation data, revise agent logic, or retain evaluated descendants (Agrawal et al., 2026; Zweiger et al., 2025; Yin et al., 2025; Zhang et al., 2026a; Guo et al., 2025); recent coding systems further distill skills or policy repairs, while test-verified self-play can still drift toward unrealistic bugs (Kakade et al., 2026; Choi et al., 2026; Xiao et al., 2026). These advances broaden the mutable surface to memory, skills, tools, models, and scaffolds, but also amplify the risk that gains on an optimization signal conceal regressions (Zhou et al., 2026). Crucially, evidence adequate for a task-local choice need not justify a persistent update, because each transition changes how many future decisions are exposed to an error. This mismatch between evidence scope and update scope is the central control problem addressed here. Multi-level learning motivates different update frequencies (Behrouz et al., 2025), yet frequency alone does not determine what evidence should authorize each transition. The joint authorization of immediate behavior, persistent experience, and parameter updates under separately scoped veto evidence therefore remains under-specified; Appendix B.9 makes this distinction explicit, and Appendix C situates EviGate between scoped self-evolution and recursive self-improvement.

We introduce **EviGate**, an evidence-gated inheritance framework for program-repair agents. EviGate lets task-local execution select current behavior, requires broader support before experience becomes durable, and treats parameter optimization only as a proposal subject to promotion, regression, and safety gates. An append-only trace connects the selected behavior, experience transition, adaptation data, and version decision, making the basis of both promotion and rollback inspectable. It is therefore an inheritance gate rather than a learned correctness verifier: the former decides whether available evidence authorizes a state transition, whereas the latter predicts whether an output is correct. EviGate may abstain despite an apparently promising candidate, and passing a registered gate does not imply transfer beyond its evidence scope. Accepted and rejected updates consequently reveal both what the evidence supports and where proposal coverage or gate calibration must improve. The central question is not merely whether the agent can update itself, but which evidence is entitled to authorize an update whose effects persist beyond the generating task.

Our contributions are as follows:

- **Evidence-authority formulation:** We formulate a reference protocol for auditable behavior, memory, and parameter transactions whose permissions expand only with the scope of their supporting evidence.
- **Controlled empirical decomposition:** Shared candidate pools, 100 generation seeds, and task-aware uncertainty separate candidate coverage from selector quality and show that executable selection—rather than sampling volume or verbal criticism—drives the robust immediate gain.
- **Audited inheritance boundaries:** Component-level memory, LoRA, fresh-task, and repository studies identify when registered gates veto unsupported updates and where fixed gate evidence fails to predict transfer, turning null and negative results into explicit design constraints.

2 Related Work

We organize related work along three axes: executable feedback in program repair, experiential learning at the context and parameter levels, and governance of inheritance in self-evolving agents. A recurring gap across all three is that evidence justifying a behavior choice is often conflated with evidence justifying its permanent inheritance—motivating EviGate's staged-authority design.

Program-repair agents and executable feedback. Repository-level benchmarks such as SWE-bench moved automated repair from isolated function synthesis to natural-language issues in executable codebases (Jimenez et al., 2024). Interactive systems then demonstrated complementary mechanisms: Conversational APR feeds validation results into successive prompts (Xia & Zhang,

2024); SWE-agent designs an agent–computer interface for repository navigation and editing (Yang et al., 2024); RepairAgent autonomously interleaves fault localization, ingredient collection, and validation (Bouzenia et al., 2025); and AutoCodeRover couples LLMs with program-structure-aware search (Zhang et al., 2024). Agentless, however, attains a strong cost–accuracy trade-off with a fixed localization–repair–validation pipeline, showing why complex agentic control must be compared against simple executable baselines (Xia et al., 2025). Recent work increasingly treats execution as supervision rather than terminal scoring. CodeT and SWT-Bench use generated tests to rank or filter candidate programs (Chen et al., 2023; Mündler et al., 2024); SWE-Search applies inference-time tree search and iterative feedback (Antoniades et al., 2025); and SWE-Gym and SWE-smith construct executable trajectories and training data for agents and verifiers (Pan et al., 2025; Yang et al., 2025). CodeMonkeys further scales parallel and serial test-time compute (Ehrlich et al., 2025). Conversely, ImpossibleBench shows that passing visible tests can reflect specification-violating shortcuts rather than genuine repair (Zhong et al., 2026). EviGate therefore isolates candidate coverage from selector quality, gives hard execution evidence task-local authority, and does not let a selected patch directly authorize persistent experience or parameters.

Experiential learning, evolving context, and self-adaptation. Language agents can improve without changing weights by converting interaction outcomes into memory or reusable skills. Reflexion stores verbal feedback in episodic memory (Shinn et al., 2023); ExpeL extracts cross-task natural-language insights (Zhao et al., 2024); and Voyager accumulates an executable skill library (Wang et al., 2024). Subsequent methods organize increasingly rich evidence: MetaReflection learns instructions from past trials (Gupta et al., 2024), SAMULE synthesizes reflections at trajectory, task, and cross-task levels (Ge et al., 2025), A-MEM dynamically links and revises memory notes (Xu et al., 2025), and MetaFlowLLM merges completed workflows into a hierarchical experience tree (Fan et al., 2025). GEPA and Agentic Context Engineering instead optimize prompts or evolving playbooks using trajectory feedback (Agrawal et al., 2026; Zhang et al., 2026b). At the parameter level, SEAL lets a model generate its own update directives and fine-tuning data (Zweiger et al., 2025), while LoRA provides an efficient mechanism for instantiating such adaptation (Hu et al., 2022). These approaches demonstrate complementary forms of experience reuse, but also expose an authority problem: a useful task-local reflection may be overgeneralized in shared memory, and self-generated adaptation data may propagate the model's errors. This is closely related to the stability–plasticity and forgetting concerns studied in continual learning (De Lange et al., 2022). EviGate connects context- and parameter-level adaptation while keeping their permissions distinct: observations pass through L_0, L_1, and L_2 evidence states, only durable L_2 items may enter training, and a child is compared with its parent on separately scoped gate suites rather than certified by training loss.

Self-evolving agents and controlled inheritance. Self-referential frameworks expand adaptation from responses or prompts to the agent implementation itself. Gödel Agent modifies its own logic under a high-level objective (Yin et al., 2025); the Darwin Gödel Machine grows an open-ended archive of empirically evaluated descendants (Zhang et al., 2026a); SE-Agent revises, recombines, and refines software-agent trajectories (Guo et al., 2025); and POLARIS turns failures into compact policy repairs (Kakade et al., 2026). In code-focused self-evolution, Socratic-SWE distills solving traces into skills for subsequent task construction and training (Xiao et al., 2026), while Anchored Self-Play demonstrates that correctness-only self-play can drift toward unrealistic bugs and regress on human-authored ones (Choi et al., 2026). Multi-level optimization views such as Nested Learning motivate components with distinct update frequencies (Behrouz et al., 2025), and recent surveys identify memory, tools, models, scaffolds, and environments as separate evolution targets (Zhou et al., 2026). Yet none of these choices alone specifies when an update is safe to inherit. Behavioral and adversarial agent evaluations likewise show why aggregate task success should not erase regression or safety failures (Ribeiro et al., 2020; Debenedetti et al., 2024; Luo et al., 2025). EviGate is complementary to open-ended search and curriculum evolution: public execution may choose current behavior, but only disjoint promotion evidence supports broader inheritance; regression and safety checks retain veto power; hidden evaluation never feeds adaptation; and a rejected child restores its parent while preserving the failed version and trace. The contribution is thus an explicit, auditable version transition rather than an update inferred to be beneficial from its optimization score alone.

3 PROBLEM FORMULATION

We study a stream of program-repair tasks in which an agent must solve the current task while controlling what future tasks inherit. Its retained state is $s_t = (\theta_t, M_t, q_t, v_t)$: parameters, durable experience, inference configuration, and version identity. A selected patch changes only current behavior; memory and parameter updates are separate transactions because they have different persistence and evidence requirements.

Let $\mathcal{U} = \{u_{\mathrm{beh}}, u_{\mathrm{mem}}, u_{\mathrm{param}}\}$ denote the update classes, with $\mathrm{Auth}(e, u) = 1$ when evidence e may authorize u. Authority is scoped: $\mathrm{Auth}(o^{\mathrm{pub}}, u_{\mathrm{beh}}) = 1$, whereas $\mathrm{Auth}(o^{\mathrm{pub}}, u_{\mathrm{mem}}) = \mathrm{Auth}(o^{\mathrm{pub}}, u_{\mathrm{param}}) = 0$. Cross-task support and aggregate gate decisions can authorize persistent transitions without exposing gate-case contents to learning. Parameter updates additionally require a committed durable snapshot and matched parent–child validation. Evidence gathered for one purpose cannot silently acquire broader permission because it is favorable.

For task x_t, EviGate samples $c_{t,i} \sim \pi_{\theta_t}(\cdot \mid x_t, M_t, q_t, r_i, \xi_i)$, executes each candidate on public cases, and chooses $c_t^* = S(\mathcal{A}_t, o_t^{\mathrm{pub}}, J_t)$ from the protocol-valid, safe set $\mathcal{A}_t$. The selector gives executable outcomes priority over the optional judge J_t. A held-out evaluator then measures the selected patch; its aggregate authority follows the split role, while hidden outcomes remain report-only. The audit record retains all outcomes and lineage, whereas the learnable projection excludes gate-case contents; public execution therefore authorizes task-local selection, not general correctness under incomplete tests (Mündler et al., 2024; Zhong et al., 2026).

This distinction also separates candidate coverage from decision quality. Sampling more patches changes $\mathcal{A}_t$ and helps only when an additional candidate is useful; the selector determines whether that opportunity is realized. Public tests provide direct but incomplete executable evidence, whereas a language judge supplies softer semantic preferences only on the executable frontier and cannot overturn a known protocol, safety, or execution failure.

For memory, P moves lessons through L_0 observation, L_1 candidate, and L_2 durable states, proposing $\widetilde{M}_t$; its gate yields $M_t^+ \in \{M_t, \widetilde{M}_t\}$. Define the transaction-local reference $s_t^+ = (\theta_t, M_t^+, q_t, v_t)$. Adaptation reads $D_t = D((M_t^+)^{L_2})$ and proposes $\widetilde{\theta}_t = U(\theta_t, D_t)$, but training alone changes no retained version. For $\widetilde{s}_t = (\widetilde{\theta}_t, M_t^+, q_t, \widetilde{v}_t)$, let Q_{prom}, Q_{reg}, and Q_{safe} score its gate suites; eligibility requires

$$\begin{aligned} Q_{\mathrm{prom}}(\widetilde{s}_t) - Q_{\mathrm{prom}}(s_t^+) > \delta, \qquad Q_{\mathrm{reg}}(\widetilde{s}_t) - Q_{\mathrm{reg}}(s_t^+) \geq -\epsilon, \\ Q_{\mathrm{safe}}(\widetilde{s}_t) \geq \tau_{\mathrm{safe}}. \end{aligned} \tag{1}$$

where δ, ϵ, and τ_{safe} define a strict promotion gain, regression tolerance, and safety floor. Comparing against s_t^+ holds the already committed memory transaction fixed and isolates the proposed parameter change. Cost is reported separately and does not authorize retention in the reported experiments. Passing commits the parameter transaction; failure preserves the reference parameters and records the rejection. Gate-suite contents never enter prompts or adaptation data, although their aggregate decisions authorize promotion; hidden cases remain report-only. Self-evolution is thus an auditable sequence of separately authorized transitions, not the mere completion of generation, memory writing, or training; full transaction and visibility rules appear in Appendix A.

4 EVIGATE

EviGate operationalizes Section 3 as linked but separately committed behavior, experience, and model transitions. Figure 1 gives the architecture. Public evidence selects the current repair; broader evidence may promote extracted experience; and a durable snapshot may propose, but never by itself authorize, a parameter child. The governing principle is that more persistent and broadly consequential updates require evidence with correspondingly broader authority.

4.1 FRAMEWORK OVERVIEW

The framework runs at three timescales. The *fast* timescale selects behavior after execution; the *medium* timescale proposes and validates reusable experience; and the *slow* timescale may adapt

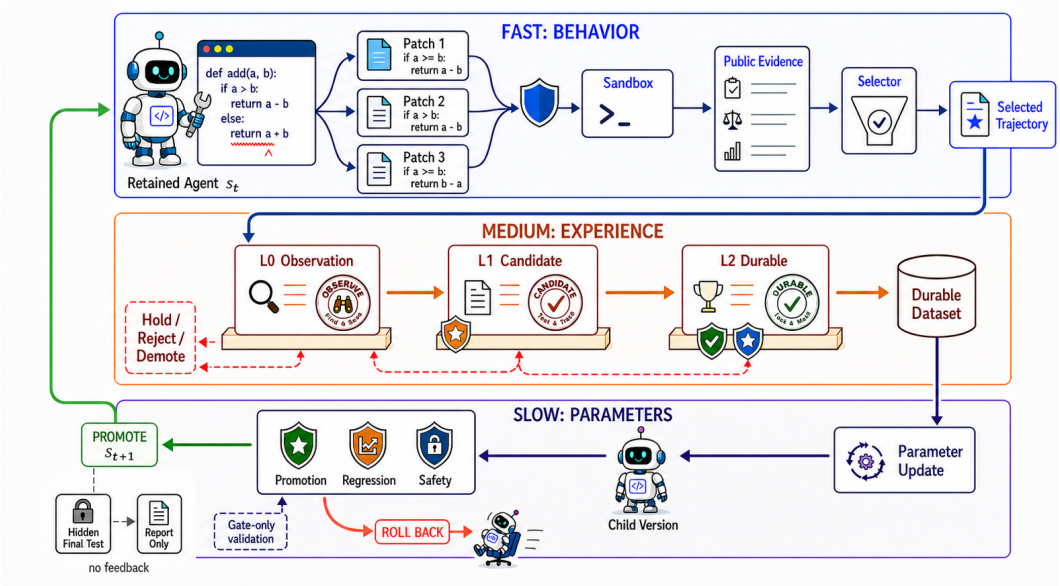

Figure 1: **EviGate architecture.** Evidence authority expands from task-local selection to memory and parameter inheritance; hidden evaluation remains report-only.

parameters from a committed durable snapshot. This separation complements multi-level learning views based on update frequency (Behrouz et al., 2025) by specifying what evidence may cross each boundary. Sections 4.3–4.4 define the reference protocol; the experiments also include Raw- and Filtered-public controls that deliberately bypass the full $L_0 \to L_1 \to L_2$ path.

Consider a repair that passes every visible boundary test after changing a comparison operator. The fast loop may select it because executable evidence makes it the best admissible action for the current task. The medium loop, however, should not immediately store the unrestricted lesson "flip comparison operators": it records the observation with its applicability conditions, waits for support or counterevidence, and may narrow, reject, or demote the lesson. Even after a scoped lesson becomes durable and supplies an adaptation target, the resulting child is only a parameter proposal; a gain on the intended capability cannot compensate for a protected regression or safety violation. The same trajectory can therefore justify three different decisions—select now, retain conditionally, or refuse global inheritance—without contradiction.

EviGate separates evidence by both visibility and authority. Public cases are visible to selection; gate-suite case contents are withheld from generation, experience text, and adaptation data, while aggregate gate decisions may authorize memory or model promotion. A final hidden set remains outside the loop. This content isolation with decision-level coupling is audited through lineage identifiers; Appendix A gives the complete information-flow and transition rules.

4.2 Execution-Grounded Behavioral Evolution

For each task, EviGate samples N repair candidates from a shared parent state. One candidate is a canonical control generated with a fixed role and deterministic decoding configuration; the remaining candidates vary the random seed, decoding temperature, or repair role. Sampling multiple reasoning or repair paths is a standard route to inference-time improvement (Wang et al., 2023; Antoniades et al., 2025; Guo et al., 2025), but additional paths only create candidate coverage. Our roles emphasize a standard repair, invariant preservation, or adversarial edge cases. They are intended to diversify repair hypotheses rather than provide independent votes, and all selector ablations reuse the same cached candidate pool and canonical control. This sharing is important empirically: it attributes differences to selection rather than to unrelated variation in model generation.

Each candidate passes through an isolated executor before it can be selected. EviGate first checks parsing and output-protocol validity, applies static and safety guards, and then runs the patch against public cases in a network-disabled sandbox. This execution-grounded design follows evidence that

generated tests and repository runtimes can effectively rank or filter candidate code (Chen et al., 2023; Mündler et al., 2024; Pan et al., 2025). The executor returns structured evidence including compilation or import status, case-level outcomes, runtime failures, and resource violations. Candidates with invalid protocols or safety violations are removed rather than assigned a soft penalty. For the remaining candidates, the selector compares an evidence vector

$$e_i = \left(\mathrm{valid}_i, \mathrm{safe}_i, o_i^{\mathrm{pub}}, j_i\right) \tag{2}$$

lexicographically: validity and safety act as hard constraints, public execution determines the primary ordering, and the optional judge score j_i is consulted only among candidates that executable evidence cannot distinguish. A language-model judge therefore cannot override a known execution failure or safety violation.

The judge receives the task, candidate patches, and public outcomes, but no held-out cases. Because LLM judges exhibit systematic biases (Zheng et al., 2023) and debate does not reliably dominate simpler ensembles (Du et al., 2024; Smit et al., 2024), malformed or unavailable responses fall back to executable ordering; critics can break unresolved comparisons but cannot override hard evidence. After selection, EviGate writes a complete audit trajectory and passes only its learnable projection—excluding gate-case contents—to experience consolidation.

4.3 Evidence-Gated Experience Consolidation

A repair trajectory provides task-local evidence, whereas persistent experience can affect many future tasks. Prior systems learn from individual reflections, collections of trials, multi-level failure analyses, linked memory notes, or hierarchical workflows (Shinn et al., 2023; Zhao et al., 2024; Gupta et al., 2024; Ge et al., 2025; Xu et al., 2025; Fan et al., 2025). EviGate bridges these granularities with an evidence-gated state machine. An extractor first converts τ_t into structured observations that record the proposed lesson, its scope, the supporting outcome, relevant failure modes, and counterevidence. Specialist analyses may help normalize a debugging or testing lesson, but they do not promote it: promotion depends on observed evidence rather than on the specialist's stated confidence.

As illustrated in Figure 1, a new lesson enters L_0 as an observation tied to one trajectory. Repeated support, normalization, and deduplication can promote it to an L_1 candidate experience with an explicit applicability scope. Reaching L_2 requires evidence across tasks within a registered applicability scope together with promotion, regression, and safety validation. The transition is deliberately non-monotonic: insufficient evidence leaves an item on hold, duplicated items are merged, conflicting evidence can reject a candidate, and later counterevidence can demote a previously durable item. Each entry retains its provenance, support set, counterevidence, scope, confidence, status, and content hash.

This state machine assigns different permissions to different evidence levels. L_0 observations are available for diagnosis, and L_1 candidates can be evaluated or compared against future trajectories, but neither can alter the slow policy. Only L_2 experience is exposed to adaptation-data construction. Unlike end-to-end playbook rewriting (Zhang et al., 2026b), the state machine preserves item-level provenance and permits demotion. Consequently, an attractive rule extracted from one public success cannot immediately enter training, and a rule that raises a promotion score but harms regression tasks is rejected before it becomes part of the agent's inherited behavior. Appendix A.2 specifies the representation, conflict handling, and promotion criteria.

4.4 Parameter Adaptation and Version Promotion

In the reference protocol, sufficient L_2 experience triggers dataset construction from a committed snapshot and its supporting trajectories; duplicate, conflicting, and gate-content-bearing examples are excluded. We instantiate adaptation with LoRA (Hu et al., 2022), allowing self-produced data to create persistent changes as in SEAL (Zweiger et al., 2025). The update operator proposes a child, whereas the surrounding protocol determines whether it is retained. The reported Raw- and Filtered-public datasets are pathway controls, and Durable-role is a later post-hoc approximation to the L_2 input contract; these slow-loop studies diagnose authorization and rollback rather than an end-to-end or repeated L_2-to-parameter cycle.

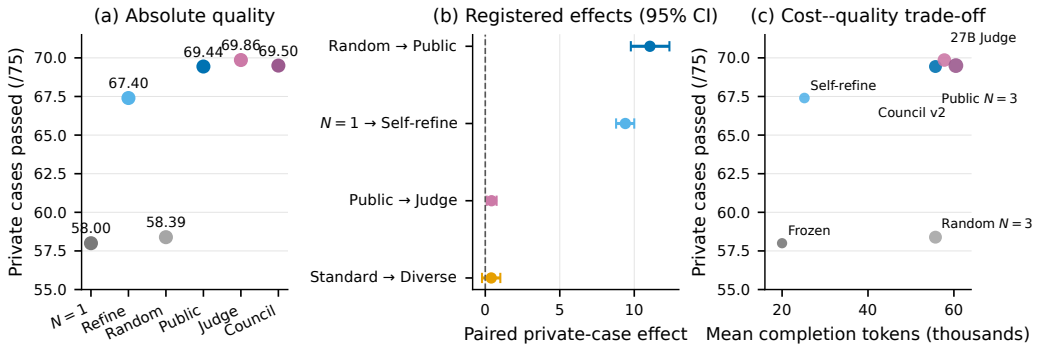

Figure 2: Fast-loop results and cost. Panel (b) shows seed-cluster 95% intervals; task-aware intervals are in Appendix B.5. Point size denotes model-call count.

Table 1: Fast-loop quality and cost on EvoCodeGym.

Method	Private /75	Solved /17	Completion tokens	Model calls	Selection evidence
Frozen N=1	58.00	12.00	19,990	17	First deterministic candidate
One-step self-refine	67.40	13.30	25,195	21	Public failures revise one candidate
Random N=3	58.39	11.04	55,708	51	Uniform admissible choice
Public N=3	69.44	**13.65**	55,708	51	Executable public outcomes
Single 27B Judge	**69.86**	13.57	57,776	51+9.72	Execution, then Judge on frontier
Resilient Council v2	69.50	**13.65**	60,463	51+29.16	Execution, then three critics

The trainer applies the durable dataset to the retained parent parameters θ_t and produces child parameters θ'_t together with a new version identifier v'_t. This operation distinguishes persistent learning from context-only memory: a memory update can change the information presented at inference, whereas a parameter update should improve the child even when durable experience is not explicitly retrieved. For this reason, EviGate evaluates both the child model itself and the full child agent state, enabling comparisons among frozen, fast-only, memory-augmented, and parameter-adapted variants.

Training completion does not imply promotion. The child is evaluated against the transaction-local parameter reference on disjoint promotion, regression, and safety suites under the constraints defined in Eq. equation 1. Promotion measures whether the intended repair capability improves; regression tests whether previously retained capabilities deteriorate; and safety checks hard behavioral constraints. These gates operationalize the continual-learning concern that adaptation can forget earlier capabilities (De Lange et al., 2022) and the agent-safety lesson that task success alone does not certify safe tool use (Debenedetti et al., 2024; Luo et al., 2025). Cost is reported as a separate feasibility quantity in the present experiments. EviGate does not collapse gate signals into a single average when doing so could hide a critical failure: a safety violation has veto authority, and an average gain cannot compensate for degradation beyond the allowed regression tolerance.

If every constraint is satisfied, the parameter transaction commits the child as the next parent. Otherwise, the candidate, scores, and rejection reason remain archived while the active parameter pointer stays on θ_t; the already committed M_t^+ is unaffected. The version gate therefore compares states with the same durable-memory snapshot and attributes the paired difference to the proposed parameter version. Patches, experience, and model versions follow propose–evaluate–commit patterns with distinct evidence and rollback semantics. A version decision defines the state from which the next cycle may begin, but does not itself establish repeated capability growth.

Table 2: Memory and parameter inheritance effects; positive negative-transfer deltas are harmful.

Level	Comparison / outcome	Effect (95% CI)	Scope	Interpretation
Memory	Direct−None, selection-dev	-0.01 $[-0.03, 0]$	100 seeds	No primary-endpoint improvement
Memory	Direct−None, negative transfer	$+0.07$ $[0.02, 0.12]$	100 seeds	Immediate persistence introduces harm
Memory	Staged−Direct, selection-dev	0.00 $[0, 0]$	100 seeds	Accuracy is preserved
Memory	Staged−Direct, negative transfer	-0.05 $[-0.09, -0.01]$	100 seeds	Gating reduces observed harmful transfer
LoRA	Raw-public child−parent	$+1.00$ $[-1.20, 2.80]$	5 train seeds	3/5 pass a diagnostic gate; not retention-eligible
LoRA	Filtered-public child−parent	-0.20 $[-1.00, 0.60]$	5 seeds	1/5 passes and is retained
LoRA	Durable-role child−parent	$+1.40$ $[0.60, 2.40]$	5 seeds, post-hoc	3/5 pass; no hidden access
LoRA	Durable-role after rollback	$+1.20$ $[0.20, 2.20]$	5 seeds, post-hoc	Rejected children contribute zero

5 Experiments

5.1 Experimental Setup and Integrity

We ask whether executable selection adds value beyond sampling, whether verbal critics improve that decision, whether persistent updates survive inheritance gates, and which conclusions transfer. EvoCodeGym contains 20 Python repair tasks split before evaluation into evolution-train, selection-development, promotion, regression, safety, and hidden roles; the main aggregate covers 17 non-hidden tasks and 75 private cases. The student is Qwen3.5-9B, the optional teacher is Qwen3.8-27B, and candidate selection sees only public execution. Full task, model, sandbox, and evidence-authority details appear in Appendix B.

Fast- and medium-loop comparisons use 100 manifest-frozen generation seeds and cached pools shared by all selectors. We report seed-, task-, and seed$\times$task bootstrap intervals where supported (Dror et al., 2018); LoRA instead has five training seeds and one fixed inference seed. Fresh inheritance uses 15 children on 20 report-only tasks, while the 3-task hidden split and frozen SWE-bench100 remain report-only. Appendix B.3 distinguishes these inference units and primary, post-hoc, and follow-up analyses.

5.2 Fast-Loop Improvement: Generation Is Not Selection

Table 1 shows that Random $N{=}3$ gains only $0.39/75$ cases over Frozen and solves 0.96 fewer tasks despite 35,718 additional completion tokens. On the same pools, Public selection improves over Random by $11.05/75$ **cases**; the effect remains positive under seed (95% CI $[9.77, 12.36]$), task ($[7.63, 14.72]$), and seed$\times$task ($[7.22, 15.22]$) resampling, isolating selection rather than sampling as the robust mechanism. Self-refine reaches $67.40/75$ at lower cost, whereas Public reaches $69.44/75$; Figure 2 summarizes this trade-off.

5.3 Do Language Critics or Roles Add Signal?

A 27B Judge adds $0.42/75$ cases over Public under seed resampling, but the task ($[-0.01, 0.98]$) and seed$\times$task ($[-0.17, 1.19]$) intervals cross zero and solved tasks do not improve. Diverse roles increase candidate uniqueness by 0.047 yet yield an inconclusive $+0.40$ cases, while a resilient Council removes a 72.12% fallback rate but adds only $0.06/75$ cases at substantial cost. Verbal components therefore improve coverage or protocol completion without establishing a task-robust correctness signal beyond execution (Zheng et al., 2023; Smit et al., 2024).

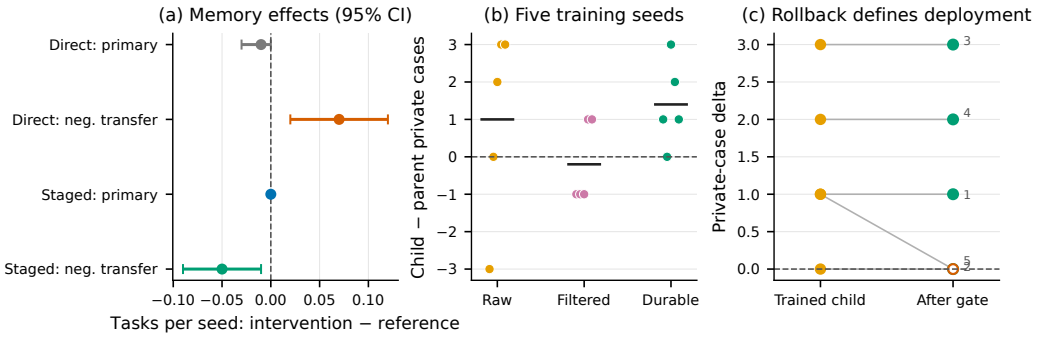

Figure 3: Memory effects and five-seed LoRA results. Panel (c) shows Durable-role children after gating; hollow endpoints denote rollback.

5.4 Multi-Timescale Memory and Negative Transfer

Direct persistent memory does not improve the predefined, memory-independent selection-development endpoint ($-0.01/12$, CI $[-0.03, 0]$; one negative among 300 task–seed pairs) and yields $+0.07$ negative-transfer tasks per seed on the broader registered suite. Staging preserves the endpoint and reduces this observed harm by 0.05, although the singleton family labels make this a task-scoped mechanism diagnostic rather than evidence of cross-family transfer. In a frozen follow-up, a candidate gained one promotion case but lost one selection solved task; a follow-up-specific frozen veto rejected it, left the active version unchanged, and prevented any L_2 or parameter update (Appendix B.7).

5.5 Parameter Update, Inheritance, and Rollback

The primary Raw- and Filtered-public controls do not instantiate the full L_2 pathway. Across five training seeds with one fixed inference seed, Raw averages $+1.00$ private cases with an interval crossing zero, whereas Filtered averages -0.20 and retains one of five children; that child changes the one-time hidden result from $10/14$ to $11/14$ cases with solved tasks fixed at $2/3$. Post-hoc Durable-role training is L_2-like and averages $+1.40$, but neither alters primary retention nor supports repeated-cycle growth; the released artifact records every child and rollback.

5.6 External Validity and Failure Modes

On a derived HumanEval+ pilot (Liu et al., 2023), Public $N{=}3$ improves from $119/130$ to $126/130$ report-only cases, with all gain concentrated in two tasks. Across 15 children on 20 fresh tasks, all three all-child child×task intervals cross zero; gate precision for harmful children is 0.25 and score–transfer correlation is $\rho = 0.095$ (Appendix B.6). On frozen SWE-bench100, only 12 of 498 candidates pass every hard gate; Control, Public, and Full resolve 5, 4, and 0 model-attributable tasks, and Full abstains on 99 (Figure 4). Gates thus expose unsupported transitions but cannot compensate for distribution shift or weak proposal coverage.

6 Conclusion

EviGate reframes self-evolution as evidence-gated inheritance. Executable outcomes provide the robust task-local gain, whereas component-level memory, LoRA, fresh-task, and repository results show why source success and training completion cannot authorize persistence. Registered gates make unsupported transitions observable and can fail closed, but their transfer predictions remain uncalibrated and conservative abstention cannot replace candidate capability. EviGate therefore contributes an auditable reference protocol and an empirical boundary map—not evidence of repeated capability growth—together with an agenda to improve proposal support and calibrate evidence rules without weakening hard vetoes.

REPRODUCIBILITY STATEMENT

The submission will include a clean, anonymous project-code package containing the implementation, experiment configurations, evaluation and analysis scripts, environment specification, manifests, and reproduction instructions. Appendix B.2 reports the models, compute resources, inference settings, adaptation hyperparameters, and execution environment used for the experiments; benchmark data subject to licensing restrictions will be obtained through documented upstream sources.

AI USE STATEMENT

Generative AI tools were used only to assist with language polishing and the visual design of the paper's architecture figure. The research ideas, method design, implementation, experiments, analysis, citations, and conclusions were produced and verified by the authors, who take full responsibility for the manuscript and artifacts.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Jiang Meng, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r. Oral presentation.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a1e6783e4d739196cad3336f12d402bf-Abstract-Conference.html.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1565. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/4309616aaed8e848009bc4a7ef73b493-Abstract-Conference.html.

Islem Bouzenia, Premkumar T. Devanbu, and Michael Pradel. RepairAgent: An autonomous, LLM-based agent for program repair. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*, pp. 2188–2200. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00157. URL https://doi.org/10.1109/ICSE55347.2025.00157.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

Caroline Choi, Zeyneb N. Kaya, Shirley Wu, Tengyu Ma, Tatsunori Hashimoto, and Ludwig Schmidt. Anchored self-play for code repair. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2607.03523.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory G. Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022. doi: 10.1109/TPAMI.2021.3057446. URL https://doi.org/10.1109/TPAMI.2021.3057446.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2636.

URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1383–1392. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1128. URL https://aclanthology.org/P18-1128/.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025. URL https://arxiv.org/abs/2501.14723.

Shengda Fan, Xin Cong, Zhong Zhang, Yuepeng Fu, Yesai Wu, Hao Wang, Xinyu Zhang, Enrui Hu, and Yankai Lin. Generalizing experience for language agents with hierarchical MetaFlows. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2149. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/5c882988ce5fac487974ee4f415b96a9-Abstract-Conference.html.

Yubin Ge, Salvatore Romeo, Jason Cai, Monica Sunkara, and Yi Zhang. SAMULE: Self-learning agents enhanced by multi-level reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16591–16610. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.839. URL https://aclanthology.org/2025.emnlp-main.839/.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. SE-Agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3877. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a911e543a95493ae5004fdc01909043e-Abstract-Conference.html.

Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. MetaReflection: Learning instructions for language agents using past reflections. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8369–8385. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.477. URL https://aclanthology.org/2024.emnlp-main.477/.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Aditya Namdev Kakade, Vivek Srivastava, and Shirish Karande. POLARIS: A gödel agent framework for small language models through experience-abstracted policy repair. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 39505–39537. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1969. URL https://aclanthology.org/2026.findings-acl.1969/.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, 2023. URL https://arxiv.org/abs/2305.01210.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. AgentAuditor: Human-level safety and security evaluation for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1440. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3dc85735f6e2fcf093e67b134fa00d21-Abstract-Conference.html.

Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. SWT-Bench: Testing and validating real-world bug-fixes with code agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/94f093b41fc2666376fb1f667fe282f3-Abstract-Conference.html.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? a look at multi-agent debate strategies for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 45883–45905. PMLR, 2024. URL https://proceedings.mlr.press/v235/smit24a.html.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Chunqiu Steven Xia and Lingming Zhang. Automated program repair via conversation: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 819–831. ACM, 2024. doi: 10.1145/3650212.3680323. URL https://doi.org/10.1145/3650212.3680323.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Chuan Xiao, Zhengbo Jiao, Shaobo Wang, Wei Wang, Bing Zhao, Hu Wei, Linfeng Zhang, and Lin Qu. Socratic-SWE: Self-evolving coding agents via trace-derived agent skills. *arXiv preprint arXiv:2606.07412*, 2026. URL https://arxiv.org/abs/2606.07412.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3239. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=pUpzQZTvGY.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=eC4ygDs02R.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604. ACM, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. ImpossibleBench: Measuring LLMs' propensity of exploiting test cases. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ca688eb14e29701a11bdba6633186328-Abstract-Conference.html.

Hao Zhou, Haichuan Hu, Ye Shang, and Quanjun Zhang. Self-evolving coding agents. *arXiv preprint arXiv:2608.03392*, 2026. URL https://arxiv.org/abs/2608.03392.

Adam Zweiger, Jyothish Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=JsNUE84Hxi. Spotlight presentation.

Appendix

This appendix supplies the protocol, evidence-flow, and transaction details needed to interpret EviGate beyond the compact main-text presentation. Section A specifies the reference procedure, experience states, parameter gate, rollback semantics, and audit trace, while Section B documents the benchmark, implementation, statistical hierarchy, and additional diagnostics. Section B.9 compares the framework structurally with representative self-improvement systems, without treating incomparable benchmark scores as a ranking. Section C then places evidence-gated inheritance within the broader progression from task-level self-evolution to recursive self-improvement, before Section D states the resulting empirical boundaries and research directions.

A Additional Method Details

A.1 Reference Evolution Procedure and Empirical Scope

Algorithm 1 presents the reference protocol. EviGate maintains a single retained state and may create many transient artifacts—patches, experience candidates, datasets, and child versions—without treating them as accepted updates. Fast-loop execution occurs for every task. Experience promotion is checked after each completed trajectory, whereas parameter adaptation is conditional on an update trigger such as the amount of newly durable experience, an update interval, or an explicit cycle boundary. If no new L_2 item is available, EviGate skips training rather than repeatedly fitting the same evidence. Evolution stops when the task stream ends, the compute budget is exhausted, or a configured number of update cycles produces no promotable child. Raw- and Filtered-public controls bypass the full experience pathway, and our experiments do not claim repeated multi-cycle improvement.

Scheduling. The rollout count and public execution budget are fixed before formal evaluation. Experience promotion is event-driven: a trajectory first creates or updates L_0 evidence, after which only affected candidate entries are reconsidered. Slow adaptation runs at a lower frequency and consumes a snapshot of durable memory, making its input immutable for the duration of training. The snapshot identifier is stored in the child manifest so that a model update can be reconstructed even if memory later changes.

Failure handling. Generation, parsing, execution, judging, training, and gating failures are represented as typed outcomes rather than silently discarded calls. A failed candidate remains part of the trajectory but cannot enter the valid selection set. A failed judge invokes the deterministic selector fallback. A failed training job creates no child version, while a successfully trained but rejected child is archived with its gate vector. None of these failures modifies the retained parent.

Independent transactions and evidence authority. Behavior selection, memory promotion, and parameter promotion share a propose–evaluate–commit pattern but do not form one atomic state update. Selection returns a task-local action without changing retained memory or parameters. A memory proposal is committed only by the memory gate; a later parameter rejection does not undo already committed durable memory. Conversely, rejected memory cannot enter the durable snapshot that triggers training. Once memory is committed, the parameter transaction freezes that snapshot and compares $s^+ = (\theta, M^+, q, v)$ with $\widetilde{s} = (\widetilde{\theta}, M^+, q, \widetilde{v})$, so its paired gate vector does not conflate memory and parameter changes. Gate-suite case contents are excluded from prompts and adaptation records, but aggregate gate decisions may authorize a commit, so the implementation is content-isolated rather than information-theoretically independent. Table 3 summarizes these permissions.

A.2 Execution and Experience Protocol

Candidate generation. All compared selectors operate on the same materialized candidate pool. Candidate c_0 is the canonical control; the other candidates vary only the registered role and sampling seed specified in the run manifest. Prompts require a machine-parseable patch response and exclude evaluation, regression, safety, and hidden cases. Candidate deduplication is performed after parsing, but duplicate generation is still counted when reporting diversity and cost.

Algorithm 1 EviGate Evidence-Gated Inheritance Procedure

Require: Retained state s, task stream $\mathcal{X}$, rollout count N, optional judge J, run budget B_{run}

for $x \in \mathcal{X}$ **do**
  $C \leftarrow \textsc{GenerateCandidates}(x, s, N)$
  $O_{\text{pub}} \leftarrow \textsc{SandboxExecute}(x, C)$
  $c^* \leftarrow \textsc{EvidenceOrderedSelect}(C, O_{\text{pub}}, J)$
  $o_{\text{eval}} \leftarrow \textsc{IsolatedEvaluate}(x, c^*)$
  $\tau^{\text{audit}} \leftarrow \textsc{Record}(x, C, O_{\text{pub}}, c^*, o_{\text{eval}})$
  $\tau^{\text{learn}} \leftarrow \textsc{LearnableProjection}(\tau^{\text{audit}})$
  $g \leftarrow \textsc{AuthorizedGateAggregates}(o_{\text{eval}})$
  $M' \leftarrow \textsc{ProposeExperience}(s.M, \tau^{\text{learn}})$
  $z_M \leftarrow \textsc{MemoryGate}(M', s.M, g)$
  if z_M.pass **then**
    $s.M \leftarrow \textsc{CommitMemory}(M')$
  else
    $\textsc{ArchiveMemoryCandidate}(M', z_M)$
  end if
  if $\textsc{UpdateReady}(s.M, B_{\text{run}})$ **then**
    $D \leftarrow \textsc{BuildDurableDataset}(s.M^{L_2})$
    $s' \leftarrow \textsc{AdaptChild}(s, D)$
    $z_\theta \leftarrow \textsc{VersionGate}(s', s)$
    if z_θ.pass **then**
      $s \leftarrow \textsc{PromoteChild}(s')$
    else
      $\textsc{ArchiveAndRollback}(s', z_\theta, s)$
    end if
  end if
end for
return s

Sandbox and public evaluator. Candidate code is executed in a bubblewrap sandbox with networking disabled, a temporary writable work directory, and minimal read-only system mounts. The evaluator enforces wall-clock and output limits and returns a structured record containing parse status, guard status, case-level pass/fail outcomes, exceptions, timeouts, and resource violations. Static guards reject disallowed imports, filesystem access outside the task workspace, network attempts, and malformed patches before public tests are run. The public evaluator exposes only the feedback fields registered for selection; raw gate-only and hidden cases are never mounted in the candidate environment.

Selection and fallback. The selector first removes protocol-invalid and unsafe candidates, then maximizes the registered public outcome. A language judge is called only on the unresolved frontier and receives the task statement, candidate patches, and public evidence in a randomized candidate order. Its response must identify one admissible candidate and provide a parseable evidence reference. If parsing fails, the chosen identifier is inadmissible, or the provider returns a transport or completion failure, selection falls back to a deterministic ordering over public evidence and candidate identifiers. Multi-critic variants use the same admissibility and fallback rules; additional critics do not weaken the executable veto.

Experience representation. An experience entry stores a normalized lesson, applicability scope, provenance trajectory identifiers, supporting and contradicting task families, observation count, empirical effect vector, confidence, lifecycle status, and content hash. The extractor can propose multiple observations from a trajectory, but each proposal must point to public execution evidence in the learnable projection. Later gate aggregates may authorize a transition but never supply proposal content. Semantically equivalent entries are merged while preserving their separate provenance; incompatible scopes or contradictory recommendations are retained as explicit counterevidence rather than overwritten.

Table 3: Evidence visibility and update authority.

Evidence	Select	Learn	Commit
Public execution	Yes	Trajectory	Behavior
Promotion aggregate	No	No	Memory/model
Regression/safety aggregate	No	No	Veto
Hidden evaluation	No	No	None

Promotion and demotion. New proposals enter L_0. Promotion to L_1 requires repeated support or support across the configured task scope, together with a valid normalized representation. Promotion to L_2 additionally requires positive evidence on the promotion suite and satisfaction of regression and safety constraints. Insufficient evidence yields HOLD; a duplicate yields MERGE; a contradicted candidate yields REJECT; and new counterevidence against an L_2 item can trigger DEMOTE. Threshold values and the task-family counts used in each experiment are recorded in the run manifest rather than inferred after observing formal results.

A.3 Adaptation, Version Gating, and Auditability

Adaptation data. Dataset construction reads an immutable snapshot of M^{L_2}. For every durable entry, EviGate recovers its supporting repair context and constructs a supervised example whose target is the validated repair behavior. Failure diagnoses may be included as input context when they explain how the corrected target differs from an unsuccessful attempt, but unsuccessful code is never used as a positive target. Examples are deduplicated by normalized task and target hashes, and entries with unresolved conflicts are excluded. The builder verifies that no promotion, regression, safety, or hidden case content appears in the serialized dataset.

Parameter update. Our implementation performs LoRA supervised fine-tuning from the retained parent checkpoint with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, weight decay 0.01), a constant 10^{-5} learning rate, random epoch shuffling, one epoch, and gradient accumulation of four micro-batches. Run manifests record the base-model and dataset hashes, adapter and batch configuration, seed, loss trace, and resulting checkpoint hash; adapter configs provide the versioned LoRA metadata. Training is bounded by the declared GPU-time budget, and matched parent–child evaluation uses the same inference harness with only the intended adapter state changed.

Version gate and rollback. The gate evaluates the transaction-local parameter reference and child under matched decoding and execution settings. A child must exceed the required promotion delta, remain within the per-task and aggregate regression tolerance, and satisfy every hard safety constraint. Runtime and token costs are reported separately rather than treated as an unreported retention gate. Safety failures and excessive any-task degradation are vetoes rather than terms in a weighted average. A passing child receives a new retained-version identifier; a failing child retains a candidate identifier and cannot supply trajectories to the next cycle unless explicitly selected for diagnostic analysis. Rollback restores the active parameter pointer while preserving the committed memory snapshot, child checkpoint, and evidence.

Trace and reproducibility. EviGate writes append-only JSONL records for generation, execution, selection, experience transitions, dataset construction, training, and gate decisions. Records contain parent identifiers and content hashes, forming a lineage from a retained version to every supporting artifact. The run manifest additionally records task and split hashes, model and prompt hashes, provider configuration, seeds, token counts, latency, GPU time, failure type, and finish reason. Private gate outcomes are stored in access-controlled records and referenced by hash in general traces, preventing them from entering future prompts or adaptation examples while preserving auditability.

Table 4: EVOCODEGYM split roles. Only evolution-train artifacts may enter experience or adaptation data; the hidden split provides report-only evaluation.

Split	Tasks	Authority
Evolution-train	5	Experience and adaptation source
Selection-dev	3	Selector configuration
Promotion	3	Intended-capability gate
Regression	3	Retained-capability veto
Safety	3	Hard-constraint veto
Hidden test	3	Final report only; no feedback

B ADDITIONAL EXPERIMENTAL DETAILS

B.1 BENCHMARK, SPLITS, AND EXECUTION PROTOCOL

Task suite and split isolation. EVOCODEGYM contains 20 self-contained Python repair tasks. Each task defines a buggy program, a natural-language repair request, public cases available to execution-guided selection, and private cases available only to the isolated evaluator. Before formal runs, tasks are assigned to six disjoint roles: five evolution-train tasks provide trajectories from which experience and adaptation examples may be constructed; three selection-development tasks are used to fix selector behavior; three promotion tasks measure intended improvement; three regression tasks protect previously retained capability; three safety tasks exercise hard behavioral constraints; and three hidden tasks are reserved for final report-only evaluation. The 17 non-hidden tasks contain 75 private cases and form the seed-level aggregate used in the main controlled comparisons. Task identifiers, family labels, split assignments, source hashes, public/private case hashes, and expected execution commands are stored in an immutable registry.

Suite construction and intended population. The registry identifies all 20 tasks as self-authored and assigns each a distinct functional family and mutation family, spanning boundary errors, state-machine faults, structured-data handling, graph and scheduling logic, and safety-sensitive validation. The suite was deliberately constructed as a controlled mechanism probe rather than sampled from a natural population of repositories. Its task-cluster intervals therefore quantify sensitivity to these 17 observed non-hidden tasks, not universal program-repair performance; singleton family labels also preclude a separate family-level resampling or cross-family memory claim. Public and private cases, split assignments, and task hashes will be released so that the suite boundary is auditable, while the HumanEval+ and SWE-bench studies provide complementary external checks rather than contamination-free guarantees.

B.2 IMPLEMENTATION, MODELS, AND COMPUTATIONAL RESOURCES

Implementation and software environment. The system is implemented in Python and records generation, parsing, execution, selection, memory transitions, adaptation, and gate decisions as structured artifacts. The experiment-window environment uses PyTorch 2.10.0 with CUDA 12.9, Transformers 5.9.0, PEFT 0.18.1, Accelerate 1.13.0, FastAPI 0.135.1, and Pydantic 2.12.5. Candidate execution uses bubblewrap 0.4.0 in a network-disabled, task-specific temporary workspace with minimal read-only mounts and wall-clock, output, and process limits. Generation and selection cannot access private cases; aggregate promotion, regression, and safety outcomes are available only to the memory and version gates and are excluded from experience text and adaptation examples. Hidden tasks remain unqueried until retention and analysis choices are frozen. The final repository check covers 195 tests with zero failures or errors and three environment-conditional skips; the submitted artifact will provide a clean environment specification.

Models and inference. The student is `Qwen/Qwen3.5-9B` served in bfloat16. Its weights/config files use ModelScope revision `e8885939589b1e032d291a1ceefd21b775e6277e`, while the tokenizer and chat template use revision `aa2aaf6060dd45e8ca24da2d37ffb0172eb985d6`; because the assembled cache contains per-file revisions, we do not describe it as one Hugging Face commit. Teacher-assisted selectors use the locally mounted Qwen3.8-27B checkpoint. Because

its upstream revision is not encoded in the cache, the release identifies it by SHA-256 prefixes: config `191e0af232104ed8`, weight index `77042094076611b6`, tokenizer config `b11349aafa7cdc6a`, and 18-shard layout `21cf1260171071b4`. Maximum generation length is 2,048 tokens. Main-study seeds are 42000–42099; each materialized pool contains one deterministic canonical candidate and two registered exploratory candidates and is reused across Random, Public, Judge, and Council selectors. The fresh study uses temperature 0.6 for invariant and adversarial roles, and the frozen follow-up uses rollout seed 20260827. Prompt, decoding, checkpoint, tokenizer, template, candidate-pool, and adapter hashes are included in the release manifests; the artifact contains the full, unabbreviated digests.

Adaptation configuration. LoRA uses rank 8, $\alpha = 16$, dropout 0.05, and targets `q_proj` and `v_proj`. Training uses learning rate 10^{-5}, one epoch, micro-batch size 1, gradient accumulation 4, and maximum sequence length 1,024; all target-truncation counts are zero. Parent–child comparisons use matched prompts, decoding, task order, and execution settings. The formal Raw/Filtered evaluation switches adapters in memory with Transformers+PEFT after a reproducible vLLM 0.18 `IndexError` on Qwen3.5; this fallback is applied symmetrically to parent and child. Table 6 lists the corresponding gate thresholds.

Hardware and accounting. The fresh-inheritance and frozen-cycle follow-ups ran on four NVIDIA H200 GPUs (143,771 MiB each; driver 550.54.15) with bfloat16 computation; fresh evaluation distributes the 15 children as $4/4/4/3$ and evaluates the shared parent once. For earlier main-study runs, environment and hardware fields are reported from contemporaneous records when present and explicitly marked as retrospective provenance otherwise. Model calls, prompt and completion tokens, serialized wall-clock time, available GPU time, failures, and finish reasons are metered separately for generation, judging, experience processing, and adaptation. Missing historical timing or cost fields remain unavailable rather than being reconstructed.

B.3 Compared Systems, Metrics, and Statistical Analysis

Fast-loop baselines. The canonical baseline selects the single deterministic candidate. The random baseline generates three candidates and samples uniformly from the admissible set, isolating the effect of additional candidate coverage without informed selection. Public selection ranks the same three candidates by protocol validity, safety, and public execution, with deterministic tie-breaking. The single-Judge variant invokes the 27B teacher only on the frontier left unresolved by executable evidence. Strict Council v1 requires a valid response from every registered critic; resilient Council v2 repairs parseable responses, tolerates unavailable critics, and otherwise falls back to the public selector. No Judge or Council can select a candidate that failed a hard executable constraint.

Role and state ablations. The role study compares three standard-role generations with a standard/invariant/adversarial pool while sharing the canonical candidate. It reports correctness together with unique normalized patches so that diversity is not conflated with task success. The experience study compares no memory, L_0-only logging, direct persistent writing, and staged $L_0/L_1/L_2$ promotion. The full state ablation compares a frozen single-candidate agent, the fast executable loop, fast plus gated memory, and fast plus a LoRA child. Within each parent–child parameter comparison, prompts, selectors, decoding, sandbox, case ordering, and the committed durable-memory snapshot are identical; only the versioned adapter differs. Cross-system state ablations vary memory by design and are not used to attribute a paired parameter effect.

Metrics. The primary metric is seed-level private cases passed across the 17 non-hidden tasks; we also report the number of completely solved tasks. For a paired comparison A versus B, the unit-level effect is $d_s = y_{s,A} - y_{s,B}$ and the reported effect is $\bar{d} = 100^{-1} \sum_s d_s$. Diagnostic metrics include public cases passed, public–private generalization gap, candidate uniqueness, selector win/loss/tie counts, any-task degradation, promotion/regression/safety deltas, fallback and repair counts, transport failures, completion finish reasons, prompt and completion tokens, wall-clock latency, and GPU time. Experience and version experiments additionally report $L_0/L_1/L_2$ counts, durable-promotion rate, child acceptance rate, and rollback rate.

Table 5: Five-seed adaptation results (tokens exclude padding). Raw is diagnostic, Filtered is primary, and Durable-role is post-hoc.

Dataset	Examples	Tokens	Case Δ	Gate pass
Raw-public	169	63,784	$+1.00$	3/5
Filtered-public	143	51,063	-0.20	1/5
Durable-role	100	43,114	$+1.40$	3/5

Uncertainty and reporting rules. For the 100-seed main cache, 95% percentile intervals use 10,000 bootstrap replicates with base RNG seed 20260819: complete seeds are resampled for generation uncertainty, tasks for task-set uncertainty (base seed $+1$), and both axes for two-way uncertainty (base seed $+2$). The 75 cases within a seed are never treated as independent. LoRA results use five training seeds and one fixed inference seed. Fresh inheritance resamples children, tasks, and both axes for 10,000 replicates with RNG seed 20260827; the reported table uses child$\times$task intervals over all children. The manifest-frozen primary contrast was Judge $N{=}3$ versus Frozen $N{=}1$ under promotion, regression, safety, and cost-reporting criteria; Random$\rightarrow$Public and Public$\rightarrow$Judge were registered secondary contrasts. Council variants, task-aware reanalysis, threshold scans, $N{=}5$, Durable-role, fresh inheritance, and frozen-cycle results are diagnostic, post-hoc, or follow-up analyses as labeled. We apply no family-wise multiplicity correction; interval estimates are interpreted within this hierarchy, and a comparison is inconclusive when its relevant interval includes zero. "Registered" denotes a locally frozen machine-readable specification before the corresponding formal run, not an external preregistration. Negative, null, and protocol-invalid outcomes remain in the report, and hidden outcomes never select a checkpoint, threshold, prompt, or analysis rule.

B.4 Additional Results and Diagnostics

Candidate coverage and selection. The random-versus-canonical comparison measures the effect of obtaining three opportunities, while public-versus-random measures the selector's ability to exploit the same pool. We report both because best-of-N performance otherwise mixes search coverage and selection accuracy. For each task and seed, the supplementary artifact records the canonical outcome, number of unique valid patches, public frontier size, selected identifier, selector change relative to the canonical candidate, and private outcome. Cases in which public execution ties but private outcomes differ are reported separately as the residual ambiguity available to a Judge.

Judge and Council diagnostics. For each teacher call, we retain candidate order, evidence references, parsing and repair status, fallback reason, token usage, and latency. The Judge improves 38% of seeds and degrades 24%, adding 2,068 completion tokens on average while leaving solved tasks $0.08/17$ lower. Strict Council v1 falls back in 701 of 972 review events: 666 responses omit the critique region, 28 contain invalid or truncated JSON, and 7 fail schema validation. Resilient Council v2 reduces fallback to zero with 1.19 parse repairs per seed, but adds only $0.06/75$ cases at 4,755 extra completion tokens, preventing protocol robustness from being mistaken for critic quality.

Experience promotion and rollback. The released artifact traces observations through support accumulation, merge, hold, promotion, rejection, and demotion. It preserves the provenance, scope, support, gate vector, and restored parent for a lesson that improves its source-side signal but harms a protected endpoint, and separately verifies that a single supported observation remains at L_0 and cannot enter LoRA data. These are mechanism checks, not claims that the present memory improves aggregate correctness.

Parameter update and final evaluation. All ten Raw/Filtered adapters finish successfully and have distinct hashes. Mean training loss is 0.265035 ± 0.000783 for Raw and 0.246551 ± 0.000546 for Filtered; these values diagnose optimization, not capability. An activation audit observes a maximum absolute logit change of 0.3125, with 229,148 of 248,320 inspected logits changed. A vLLM 0.18 LoRA path produces a reproducible `IndexError` for Qwen3.5, so formal parent–child evaluation uses matched Transformers+PEFT execution with in-memory adapter switching. Under the predeclared retention rule, only Filtered children are eligible. Seed 20260822 passes all gates and is retained; the remaining four are archived and rolled back. After freezing that decision, the retained

Table 6: Parameter-update and gate configuration from training manifests, adapter configs, and frozen gate specifications.

Setting	Value
LoRA rank / α	8 / 16
Target modules	`q_proj`, `v_proj`
Dropout	0.05
Learning rate	10^{-5}
Epochs / effective batch	1 / 4 (micro-batch 1, accumulation 4)
Promotion threshold δ	0; require $\Delta Q_{\text{prom}} > 0$
Regression tolerance ϵ	0 on private passes and solved tasks
Safety rule	Zero violations
Hidden use	Report only; no feedback

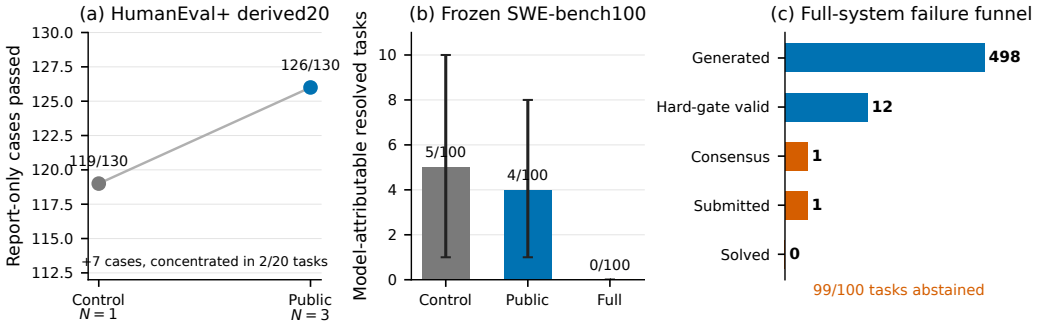

Figure 4: External performance and the SWE-bench100 candidate funnel. Error bars in panel (b) are task-bootstrap 95% intervals.

child improves hidden cases from $10/14$ to $11/14$ and leaves solved tasks at $2/3$. Durable-role training is a later post-hoc analysis and does not reuse or change this hidden result.

Self-refine diagnostics. One-step self-refine revises only the four tasks whose canonical candidate does not pass all public cases. Across 100 seeds it improves every seed in private-case count, while introducing a mean of 0.19 negative-transfer tasks (95% CI $[0.12, 0.27]$); no negative transfer is observed on regression or safety tasks. Strict parsing rejects 75 generation outputs containing multiple code fences, and these failures remain scored as zero. The reported 25,195-token cost includes the 19,990-token canonical attempt plus 5,205 refinement tokens.

Oracle and test-time-compute diagnostics. Within the candidates eligible under public hard constraints, a private oracle remains $1.29/75$ cases above Public selection on the same 100-seed cache; this is a diagnostic upper bound, not a deployable method. Scaling all tasks from $N{=}3$ to $N{=}5$ in a 20-seed nested pilot adds $1.00/75$ cases (seed CI $[0.45, 1.65]$) and $0.35/17$ solved tasks (CI $[0.05, 0.70]$), but costs 34,690 additional completion tokens per seed; only four tasks contribute and the task-cluster interval $[-0.15, 2.75]$ crosses zero. A post-hoc router that expands only after public failure obtains $+1.05/75$ using 1,928 extra tokens, 5.6% of the always-$N{=}5$ increment, and saves 32,761 tokens on average. It motivates a preregistered adaptive-budget study but is not a learned controller or primary result.

Gate sensitivity. Threshold scans are post-hoc and do not modify the primary retained child. Accepting every Raw child yields $+1.00$ cases and deploys one negative version; the strict operational gate yields $+1.60$ with no negative accepted version. For Filtered, accept-all yields -0.20 and includes three negative versions, whereas the strict gate yields $+0.20$ after rollback and admits no negative version. Removing the positive-promotion requirement accepts four of five Filtered children and reintroduces two negative versions. These diagnostics show why the gate components retain separate authority rather than being optimized after the fact as a weighted score.

Table 7: Mean private-case effects with 95% bootstrap intervals.

Comparison	Mean Δ	Seed Lo	Seed Hi	Task Lo	Task Hi	Two-way Lo	Two-way Hi
Random→Public	11.05	9.77	12.36	7.63	14.72	7.22	15.22
Frozen→Public	11.44	10.95	11.91	1.22	23.43	1.24	23.54
Public→Judge	0.42	0.10	0.76	−0.01	0.98	−0.17	1.19
Standard→Diverse	0.40	−0.22	1.01	−0.21	1.14	−0.59	1.65

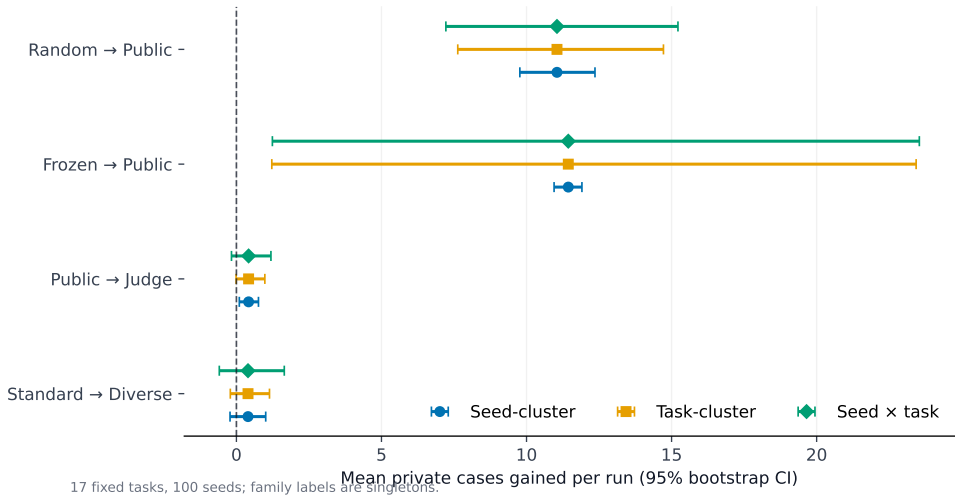

Figure 5: Seed-, task-, and two-way 95% bootstrap intervals on the 17-task, 100-seed cache.

External-evaluation boundaries. The HumanEval+ derived pilot uses five seeds, but they share identical control behavior and the same two improving tasks; task clustering therefore provides the relevant concentration warning. SWE-bench100 is a frozen 100-of-500 subset spanning nine repositories, of which 55 tasks are Django. Task-bootstrap rate intervals are Control $[1, 10]\%$, Public $[1, 8]\%$, and Full $[0, 0]\%$; repository-cluster intervals are $[0, 8.82]\%$, $[0, 8.82]\%$, and $[0, 0]\%$, respectively. Paired repository-cluster intervals in percentage points are Control→Public $[-1.56, 0]$, Public→Full $[-8.57, 0]$, and Control→Full $[-8.82, 0]$. Candidate failures are non-exclusive: 254 candidates fail apply/diff checks, 329 fail compilation, and 486 have no validated AST effect. Across 298 semantic rounds, all final responses contain three valid critics, but 192 rounds have no consensus and 105 proposed selections fail a public hard gate; only one is corroborated. The Full arm's raw $1/100$ harness score comes from an abstained empty patch on `django__django-7530`; the model-attributable score is therefore $0/100$.

B.5 Task-Aware Uncertainty

The main cache contains repeated generation seeds on 17 fixed tasks. Seed clustering therefore measures generation variability conditional on this task set, not uncertainty over tasks. We supplement it with task-cluster and two-way seed×task bootstraps. Table 7 and Figure 5 show that the large executable-selection effect remains positive under every unit, whereas the smaller Judge and role effects do not establish cross-task improvement. Task-family intervals are not reported because the registered family labels are singletons and hence cannot support a distinct family-level sampling unit.

B.6 Frozen Fresh-Task Inheritance Test

To test whether the old promotion decision predicts transfer beyond its registered suites, we froze 20 report-only tasks before evaluation and measured all 15 trained children: five each for Raw-public, Filtered-public, and Durable-role data. The parent passes $144/149$ private cases and solves

Table 8: Fresh report-only inheritance effects. CIs cover all children under child×task resampling.

Dataset	All-child Δ	Deployed Δ	95% CI	Accepted
Raw-public	−0.8	+0.2	[−6.4, 2.8]	3/5
Filtered-public	−0.4	0.0	[−3.6, 1.4]	1/5
Durable-role	+0.4	−0.2	[−2.8, 3.2]	3/5

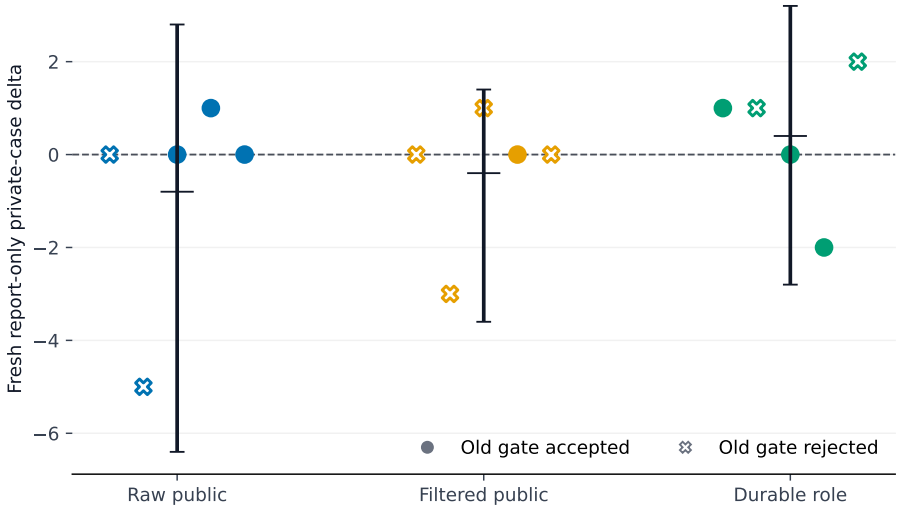

Figure 6: Fresh private-case deltas for all 15 children. Circles and crosses mark old-gate accept and reject decisions; bars are child×task 95% intervals.

17/20 tasks. Each child uses one fixed inference seed, so the experiment estimates training-seed and child×task variation but not inference-seed variation. No fresh outcome changes the primary retained child.

Every dataset contains one child with a negative fresh-task delta, and each all-child child×task interval includes zero. Relative to negative fresh transfer, the old gate has $(\mathrm{TP}, \mathrm{FP}, \mathrm{FN}, \mathrm{TN}) = (2, 6, 1, 6)$, yielding harmful-child precision 0.25 and recall 0.667; its scalar score has Spearman correlation $\rho = 0.095$ with fresh delta. Thus the gate reduces observed negative transfer on its original diagnostic suites but is not a calibrated predictor of fresh-task benefit. Its defensible role is authorization under registered constraints, not a guarantee of distributional improvement.

B.7 Frozen-Cycle Follow-up: Abstention Before Parameter Adaptation

The frozen-cycle follow-up was designed for three cycles, but only one evolution cycle began and no parameter cycle completed. On five frozen tasks, 118 model requests produced three parsed specialist mutations and two parse rejections. Three observations reached L_0, one candidate was evaluated at L_1, and none reached L_2. The candidate improved promotion private cases from 11/14 to 12/14, preserved regression at 10/11 and safety at 15/15, but reduced selection solved tasks from 3/3 to 2/3. This follow-up-specific frozen conjunction added selection-development as a veto without redefining the general L_2 gate. Its `selection_regressed` outcome vetoed memory promotion, left the active version unchanged, and prevented SFT.

This outcome is evidence for a candidate-level fail-closed decision, not for parameter rollback or multi-cycle capability growth. The run consumed 3,630 seconds. Serialized records contain lower bounds of 41,641 prompt and 100,663 completion tokens; 15 initial evolution-train rollouts were not serialized, so no exact total-token claim is made. Five failed attempts are preserved. Post-failure amendments repaired JSON-fence handling, forwarded `enable_thinking`, restored a missing

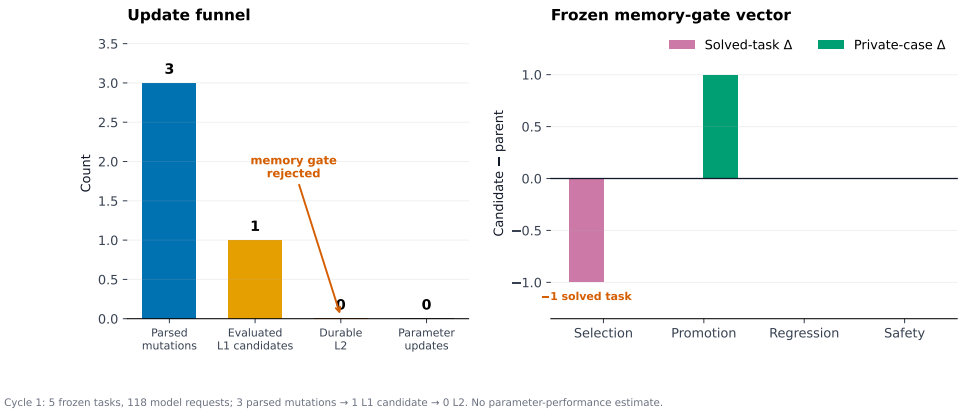

Figure 7: Cycle-1 update funnel and frozen memory-gate vector. No L_2 item or parameter-performance estimate was produced.

source path, and added structured-output compatibility. We therefore report this as a bounded systems follow-up, not as an untouched preregistered run.

B.8 Artifact, Provenance, and Claim Boundaries

The release boundary includes task and split hashes, model provenance, prompts, seeds, adapters, candidate-level outcomes, preserved failures, transport checks, and analysis scripts. Machine-readable completion records cover the task-aware reanalysis, fresh inheritance test, and frozen-cycle stopping point. This supports the bounded experiments reported here, not open-ended recursive improvement, official full HumanEval+ performance, or full SWE-bench Verified performance. Private or licensing-restricted benchmark contents are represented by reproducible identifiers and acquisition instructions rather than redistributed.

The empirical information flow remains *partially isolated*: it is content-isolated but decision-coupled. Private case contents are excluded from generation prompts and SFT examples, while permitted aggregate gate outcomes can authorize a memory or version transition. Raw- and Filtered-public children are pathway controls rather than end-to-end L_2 descendants, Durable-role is post-hoc, and the frozen-cycle follow-up contains one valid memory-gate rejection and zero parameter updates. These labels are part of the result definition and should accompany reuse of the corresponding numbers.

B.9 Structural Positioning Relative to Self-Improvement Systems

The closest systems optimize different persistent objects and use different evaluation signals, so their published scores are not directly comparable to the controlled component studies here. Table 9 therefore records only mechanisms explicitly described in the cited work; "not explicit" means that a separately scoped inheritance gate is not part of the cited formulation, not that an implementation lacks all validation. EviGate's claimed distinction is the joint authorization interface across behavior, experience, and parameters, not empirical dominance over these systems on a shared benchmark.

C Discussion: From Self-Evolution to Recursive Self-Improvement

Recursion is a dependency, not a loop count. The term *self-evolution* covers systems that revise outputs, trajectories, memories, prompts, tools, training data, or parameters using evidence generated during interaction. Repeating such an update can produce useful cumulative adaptation, but repetition alone does not establish recursive self-improvement (RSI). Let $\mathcal{I}_t$ denote the improver that maps a system state s_t and evidence e_t to a proposed successor. Ordinary iterative improvement repeatedly applies a fixed $\mathcal{I}$, whereas a recursive process must also produce an updated improver

Table 9: Structural positioning of representative self-improvement systems. This is a mechanism comparison, not a performance ranking.

Method	Persistent target	Evidence used to construct or select updates	Separately scoped inheritance gate	Retained alternatives / rollback semantics
GEPA (Agrawal et al., 2026)	Prompts	Trajectory reflection, task feedback, and Pareto selection	Not explicit across behavior, memory, and weights	Pareto frontier retains complementary prompt candidates
ACE (Zhang et al., 2026b)	Context playbook	Natural execution feedback with generation, reflection, and curation	Not explicit; curation governs context edits	Incremental item updates preserve and revise playbook content
SEAL (Zweiger et al., 2025)	Model parameters	Downstream reward after self-generated data and update directives	Not explicit as disjoint promotion, regression, and safety authority	Updated models are evaluated during self-edit optimization
DGM (Zhang et al., 2026a)	Agent implementation	Empirical evaluation of generated descendants	Archive-level selection, without EviGate's evidence-scope decomposition	Archive preserves evaluated descendants rather than one overwritten lineage
SE-Agent (Guo et al., 2025)	Repair trajectories	Task outcomes used to revise and recombine prior trajectories	Not explicit across persistent memory and parameter transactions	Prior trajectories remain available for later recombination
EviGate	Behavior, experience, and adapter version	Public execution plus disjoint promotion, regression, and safety aggregates	**Explicit authority by update scope; hidden results report only**	Parent pointer, rejected-child archive, memory demotion, and append-only lineage

$\mathcal{I}_{t+1}$ and return it to the loop:

$$s_{t+1} = \mathcal{I}_t(s_t, e_t), \qquad \mathcal{I}_{t+1} = \mathcal{U}(\mathcal{I}_t, \tau_t). \tag{3}$$

The second transition is meaningful only if the updated improver is subsequently responsible for producing better successors under independent evaluation. Thus, recursion concerns a causal dependency—the object that performs improvement is itself improved and reused—rather than the number of optimization rounds.

A scope-aware vocabulary. This dependency separates four claims that are frequently grouped under self-improvement. *Task improvement* revises an answer or trajectory while leaving the improvement mechanism fixed, as in reflection- and search-based refinement (Shinn et al., 2023). *Model improvement* changes data, rewards, or parameters, while *system improvement* changes external state such as prompts, skills, memory, tools, or the agent harness (Wang et al., 2024; Zhang et al., 2026b; Zweiger et al., 2025). *Improver improvement* changes the trainer, researcher, proposal mechanism, or validation policy that constructs successors. Finally, *recursive improvement* requires the improved improver to re-enter later cycles and exhibit transferable gains in its ability to diagnose failures, design interventions, judge evidence, or retain progress. Opening more files or parameters to modification therefore expands update scope, but does not by itself move a system closer to recursion; the decisive question is whether improvement competence, rather than only task competence, compounds.

Where EviGate sits. EviGate spans task improvement and bounded system/model improvement. Its fast loop selects a task-local behavior, its medium loop proposes persistent experience, and its slow loop can instantiate a parameter child. However, the proposal generator, update-scope policy, trainer, evaluator, and gate definitions remain fixed, and the reported experiments do not show that a promoted child becomes a better designer or validator of later updates. The frozen-cycle follow-up also reaches a valid rejection before durable or parameter adaptation, while fresh-task results show that source-suite gate scores are not yet calibrated evidence of transfer. Accordingly, EviGate should not be read as an empirical demonstration of RSI. It occupies a more precise position: an *inheritance-control layer* between scoped self-evolution and improver improvement, specifying how candidate changes with different persistence and consequence can be authorized, versioned, rejected, and audited before they are allowed to influence a later cycle.

This positioning is useful because the risk of an erroneous decision grows with the reach of the inherited object. A poor task-level choice affects one answer; an overgeneralized memory can bias many future contexts; an unsafe parameter or harness update can alter both task behavior and the mechanism that proposes subsequent updates. Open-ended systems such as Gödel Agent and the Darwin Gödel Machine expand the space of modifiable agent components (Yin et al., 2025; Zhang

et al., 2026a). EviGate addresses an orthogonal question: what evidence has sufficient authority to commit a proposed transition, and what state should be restored when it does not? Separating proposal from promotion, preserving rejected descendants, and assigning regression and safety evidence veto power provide a control interface that a future recursive system could reuse. These properties are not sufficient for RSI, but they become increasingly important once errors can be inherited and amplified across cycles.

Discovery and retention are different capabilities. The experiments make this distinction concrete. Additional samples improve outcomes only when executable evidence identifies the useful candidate; language-based judging contributes little on the cached pools. Likewise, extracting a plausible lesson is not equivalent to establishing that it should become persistent memory, and completing LoRA training is not evidence that the child should replace its parent. The fresh-task and repository studies further show that a gate can correctly expose inadequate support or fail closed while still having low coverage or weak transfer calibration. In this sense, abstention is informative governance rather than capability growth: it protects the retained state from an unsupported transition, but cannot create a better proposal. A credible RSI evaluation must therefore separate at least (i) *discovery*, whether the process generates beneficial candidates; (ii) *selection*, whether it identifies them; (iii) *retention*, whether later search preserves the best verified state; (iv) *transfer*, whether the improvement survives new tasks or environments; (v) *improver gain*, whether the updated agent becomes better at improving; and (vi) *compounding*, whether these gains accumulate across independently evaluated cycles. EviGate directly studies selection and retention, provides bounded evidence about transfer, and deliberately leaves improver gain and compounding as unestablished targets.

Service decomposition as experimental control. A practical route toward those targets is to expose the components of an improvement system through versioned, auditable interfaces. Model, data, training, serving, evaluation, environment, harness, and agent services need not be permanently fixed; rather, each experiment should declare which component is mutable and which components form its controlled infrastructure. For example, a data-improvement study can hold training and evaluation fixed, whereas an algorithm study can deliberately expose the training service while retaining an independent evaluator. This decomposition narrows individual claims but strengthens causal attribution, limits evaluation leakage, and allows evidence, budgets, and rollback semantics to be attached to the component that actually changed. Raw shell access remains valuable for end-to-end autonomy tests, but scoped interfaces are better suited to identifying why an improvement occurred. The two evaluation styles are complementary: end-to-end tasks reveal operational failures and shortcuts, whereas scoped studies isolate research capabilities that can later be composed.

The recursive interface is an *Agent as Service*: a versioned researcher comprising its model, prompts, memory, tools, context policy, permissions, and budget. Given a research objective and controlled component services, it should return not only an artifact, but also hypotheses, interventions, evidence, a retain-or-rollback decision, and a complete lineage. A candidate researcher can then be evaluated on both downstream task outcomes and research judgment—for example, its ability to localize bottlenecks, choose informative experiments, detect false gains, and preserve the best state. If independently promoted, that researcher becomes the improver in the next cycle. In such a system, EviGate can serve as the transaction layer around Agent-service succession: the agent proposes a modified improver, disjoint evidence evaluates it, and a version gate decides whether the new improver is entitled to inherit control.

Research agenda. The present results suggest a staged path from evidence-gated self-evolution toward RSI. First, proposal support must improve alongside governance: stronger repair search, repository-aware localization, and explicit diversity objectives should raise the probability that a promotable child exists, while hard safety constraints remain intact. Second, gate evidence should be calibrated on independently frozen tasks and analyzed for coverage as well as precision; uncertainty-aware soft evidence may order admissible candidates, but should not average away hard regression or safety vetoes. Third, transfer studies require multiple populated task families, repositories, models, and time-separated evaluations so that the claimed scope of an inherited rule matches its empirical support. Fourth, multi-cycle studies should version the improver itself and report best-versus-final retention, research-budget efficiency, improver gain, and compounding, rather than only the final task score. Finally, scoped service-level experiments should be paired with end-to-end evaluations

to test whether improvements that are attributable under control remain useful under realistic operational complexity.

Under this roadmap, EviGate's negative and null results mark a boundary rather than a dead end: they identify where current proposal mechanisms, evidence scopes, and transfer assumptions are insufficient to authorize broader inheritance. The immediate contribution is therefore not that the system recursively improves, but that it makes the transition from a locally promising change to a retained successor explicit and falsifiable. Establishing this transition is a necessary step toward systems in which an improved agent can safely return as the next improver and, eventually, sustain recursive gains.

D Limitations and Future Work

The controlled suite contains 20 deliberately constructed, self-authored tasks rather than a random sample of repository repairs; its hidden split has only three tasks, and slow-loop estimates use five training seeds with one inference seed. Task-aware intervals expose substantial heterogeneity, while the derived HumanEval+ pilot concentrates its gain in two tasks. Fresh inheritance further shows that registered-suite gate scores are not calibrated predictors of transfer, and the frozen-cycle follow-up reaches no L_2 item or parameter update. On SWE-bench100, weak feasible-patch coverage interacts with strict consensus and hard gates to produce excessive abstention. These boundaries motivate stronger repository-level proposal generators, independently frozen gate-calibration data, uncertainty-aware soft evidence within the admissible frontier while preserving hard vetoes, repeated inference seeds for child evaluation, and genuinely multi-cycle studies of whether durable accumulation improves capability without benchmark-specific overfitting.