Can Agent Improvements Survive Repeated Selection?

Fresh, capability-stratified promotion tests for self-modifying coding agents

Abstract

An agent that repeatedly modifies its own harness can overfit the tests used to select successors, even when its evaluator is read-only. We propose a promotion protocol that evaluates one nominated candidate per generation on a fresh, sealed batch of tasks and imposes capability-specific noninferiority requirements. We will compare the protocol with static regression gates under equal search and test budgets, measuring whether apparent improvements transfer to a separate terminal test and how many useful edits are lost to conservative gating.

Keywords / recursive self-improvement, coding agents, adaptive selection, regression testing

Problem and prior work

The Darwin Godel Machine evolves a population of agent variants [1], while recent research-agent RSI work measures transfer to held-out benchmarks [2]. Capability-Preserving Evolution explicitly constrains forgetting during agent adaptation [3]. Our question concerns repeated adaptive exposure to a promotion signal. Even pass/fail outcomes can shape later mutations, so a reused regression suite may gradually stop forecasting performance on untouched tasks. The study tests task refresh while holding the preservation objective constant.

Hypothesis and method

Freeze the base model. Let the incumbent propose edits to its own planning, tool routing, or patch verification configuration and run cheap public development tasks. The incumbent nominates one candidate before an external evaluator draws a sealed batch. Promote only if the paired improvement on target tasks is positive by a prespecified confidence rule and no protected capability exceeds its noninferiority margin. Return a pass/fail bit, keep a versioned lineage, and use the surviving incumbent to write the next round of edits. Test and evaluator directories are isolated from the agent.

Statistical scope

Use predeclared, per-round error spending and stratified paired intervals. Task refresh addresses feedback from reused instances. It cannot remove dataset-family bias, so terminal evaluation also holds out entire task generators or repositories. Compare both power and false promotions using injected near-null and deliberately regressive variants.

Validation plan

Experimental design

Run up to ten generations with five independent lineages per condition. Use executable repair tasks in capability strata such as parsing, state handling, and API integration, grouped by repository or generator family. Budget a starting maximum of 120 paired promotion evaluations per generation and choose attainable margins after a disjoint pilot. Compare dev-score-only promotion, static protected tests, fresh aggregate tests, and fresh stratified tests. Give each arm the same candidate proposal and evaluation budgets. Evaluate selected incumbents on a sealed terminal pool whose scores are revealed after all lineages finish.

Outcomes and rejection criteria

The primary outcomes are terminal target-task gain and the worst protected-capability change relative to the seed. Secondary outcomes are promotion-to-terminal score optimism, false promotions, false rejections, and cost per retained gain. Report uncertainty across lineages and repository clusters. The protocol is supported only if it improves terminal capability retention against the static stratified gate while preserving measurable target gains. If the gate blocks most candidates, report the attainable trade-off and test whether limited statistical power explains the result.

Feasibility and deliverables

Weeks 1-2 prepare the task partitions and reproducible runner. Weeks 3-4 calibrate the gate on known variants. Weeks 5-7 run lineages and unlock the terminal evaluation. Use disposable offline sandboxes, fixed timeouts and an immutable evaluator. Release lineage logs, task-generator configurations, and a reproduction harness. The claim concerns harness-level RSI over a bounded number of generations.

References

[1] Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents. arXiv:2505.22954.

[2] Recursive self-improvement of AI research agents. arXiv:2609.26457.

[3] Do Self-Evolving Agents Forget? Capability Degradation and Preservation in Lifelong LLM Agent Adaptation. arXiv:2605.09315.