COUNTERGRAFT: COUNTERFACTUAL EXECUTION FOR AGENT SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

A higher-scoring agent can still make worse behavioral choices: stronger downstream execution can conceal unnecessary or harmful changes. We introduce **CounterGraft**, a method for improving executable agent policies and prompts through counterfactual feedback. Its core operation runs the successor from the incumbent's selected choice and execution state, then compares the outcome with execution from the successor's own handoff. Holding the successor policy fixed reveals whether the replacement improves declared utility, supplying revision feedback unavailable from paired endpoints alone. CounterGraft uses this feedback to repair programs, then independently audits the selected program before release. On AgentDojo, it improves held-out task success by up to 32.9 percentage points over incumbent agents and by 1.3–1.9 points over the strongest adapted baseline across three executors; it also achieves the highest success among the compared methods on ALFWorld. In a matched AgentDojo experiment, exposing counterfactual feedback reduces harmful replacements by 70% while increasing task success, with the proposer, executor, and proposal budget fixed. Repairs recover required actions and improve replacement quality at an unchanged departure count. The release audit controls false certification on the bound workload under adaptive stopping, reduces replay beyond deterministic checks, and enables certified successors to become subsequent incumbents in prospective runs.

1 INTRODUCTION

An agent can improve without changing its model weights. Revising the prompts, tool-use policies, and control logic around a frozen model can change which information it acquires, which actions it takes, and when it stops. Recent methods automate this process by generating and evaluating new agent programs (Hu et al., 2025; Zhang et al., 2025; Zhou, 2026). This makes the feedback supplied to a revision algorithm consequential: it must help the proposer decide which changes to retain, condition, or remove. Endpoint scores and execution traces describe how the new program performed, but do not directly measure what each replacement contributed under that program. Optimizing the overall score can therefore preserve changes that add cost or cause failures even as the successor improves.

The difficulty is that a successor changes both its choices and the execution that follows them. Consider an extra read before a document update. The read may supply information required by the successor's new policy, or duplicate information already available. Successful executions of both versions leave this unresolved because the incumbent uses a different continuation. The missing evidence is the successor's outcome if it retains the incumbent-selected choice and execution state, which we call a *handoff*. Comparing this outcome with the successor's own execution holds its continuation policy fixed and reveals whether the replacement improves declared utility. In our nonparametric execution-table model, paired endpoints do not generally identify this value; one additional *child-on-parent execution* supplies the missing record.

CounterGraft makes this comparison the basis of a repair-and-release procedure (Figure 1). During repair, replacement labels and traces guide a proposer toward executable rules that retain useful changes and revise harmful or unnecessary ones. Each new program is re-evaluated because its continuation can alter the value of its choices. Before release, the selected program is frozen and subjected to an independent audit of task regressions and behavioral replacement value. The audit

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

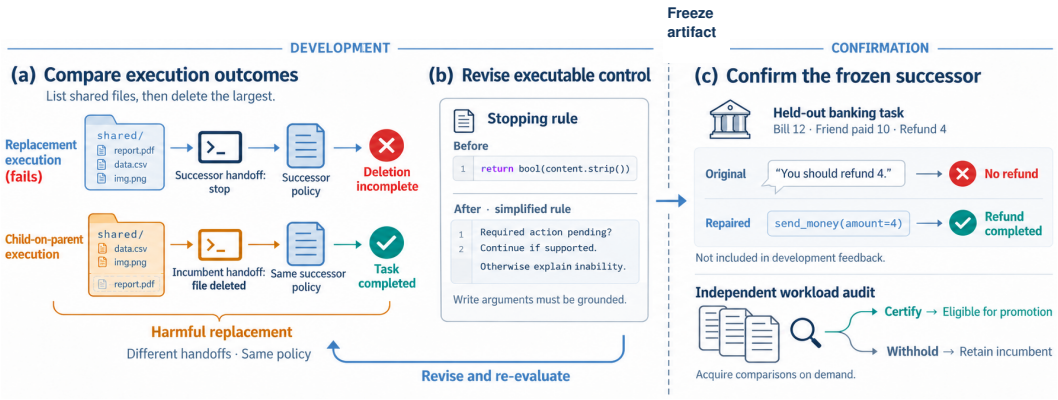


Figure 1: **From counterfactual feedback to executable repair and independent release.** A development failure guides a stopping-rule revision. The selected program is frozen before confirmation; the banking task is held out from development feedback. The audit evaluates release eligibility using newly bound records. Task text and replies are abbreviated.

acquires counterfactual outcomes only while its decision remains unresolved, combining deterministic completion checks with sequential evidence. The guarantee controls false certification on the bound workload under adaptive stopping and across adaptively proposed releases. Certified programs can then become subsequent incumbents.

With GPT-4.1 fixed as executor, CounterGraft raises AgentDojo success from 62.9% to 95.8%. Across three AgentDojo executors it exceeds the strongest adapted baseline by 1.3–1.9 percentage points, and the same performance ordering holds on ALFWorld. Eight of nine AgentDojo selections are certified, compared with five for adapted HSI. A separate matched-feedback experiment holds search and selection fixed: exposing counterfactual evidence reduces harmful replacements from ten to three while increasing success. Repair also improves replacement utility at the same departure count, learning when to act and when to stop.

Our contributions are (i) an executable, successor-relative comparison that identifies replacement value from the branch missing in paired evaluation; (ii) a repair-and-release algorithm that turns this comparison into code-revision feedback and independently acquired certification evidence; and (iii) experimental evidence connecting cross-model performance gains to better behavioral choices, with matched-feedback controls and prospective successor installation. A further 17-release panel tests the comparison and audit interface across model, component, and routing updates.

2 Counterfactual Execution for Successor Improvement

2.1 Replacement value under the candidate continuation

We compare two choices under the same candidate policy: the handoff selected by the incumbent and the one selected by the candidate. A handoff includes both the boundary choice and the execution state needed to continue. Let A_0 be the incumbent and A_1 the candidate, with selected handoffs $h_0(x)$ and $h_1(x)$ on task x. A declared projection q specifies which differences count as changed behavior: a departure occurs when $q(h_1(x)) \neq q(h_0(x))$. In AgentDojo, q compares canonical tool-call blocks or terminal actions; in ALFWorld, it compares primitive environment actions. State is retained for execution but does not itself create a departure.

Continuing version e from handoff h produces $Y_e(h) = (r_e(h), u_e(h))$. Task adequacy r_e uses a common ordered scale; utility u_e uses a declared total preorder $\succeq$, with strict preference $\succ$ and ties $\sim$. Child-on-parent execution supplies $Y_1(h_0)$. Comparing $u_1(h_1)$ with $u_1(h_0)$ labels the departure *beneficial*, *harmful*, or *tied*. The candidate policy is fixed across these branches, but its subsequent actions can respond to the retained state.

The comparison concerns complete handoffs when their histories differ; with a shared pre-decision state, it specializes to an action replacement. The execution interface fixes state restoration, action

timing, and cost accounting so that the boundary action enters exactly once. AgentDojo restores the incumbent's post-action state and resumes with candidate code (Appendix A).

2.2 THE EXECUTION MISSING FROM PAIRED EVALUATION

Suppressing x, paired evaluation observes $Y_0(h_0)$ and $Y_1(h_1)$. The additional record $Y_1(h_0)$ completes the *selected-handoff triangle*. For numerical utility on a common scale,

$$u_1(h_1) - u_0(h_0) = \underbrace{u_1(h_0) - u_0(h_0)}_{\text{continuation change at } h_0} + \underbrace{u_1(h_1) - u_1(h_0)}_{\text{replacement under } A_1}. \tag{1}$$

A positive endpoint gain can coexist with a negative replacement term. Ordered utilities use direct preference comparisons instead of subtraction.

In our nonparametric execution-table model, each executable version–handoff pair has an admissible outcome; the interface, handoffs, projection, and outcome spaces are fixed. No cross-entry relations are assumed beyond those constraints. Endpoint evidence is $O_{\text{end}} = (h_0, h_1, Y_0(h_0), Y_1(h_1))$.

Proposition 1 (Endpoint non-identification)**.** *Over this execution-table class, O_{end} does not uniformly identify whether a behavioral departure is beneficial under the candidate continuation.*

For example, hold endpoint utilities at 0.5 and 0.6. An unobserved $u_1(h_0)$ of 0.4 makes the replacement beneficial, whereas 0.8 makes it harmful; every observed field can be identical. Reobserving those endpoints cannot distinguish the worlds.

Proposition 2 (Selected-handoff sufficiency and minimality)**.** *Given the selected handoffs and projection, the three records identify both endpoint regression and the replacement label. No strict subset of these primitive queries uniformly identifies both over the nonparametric class with executable departures and nondegenerate outcome spaces.*

The candidate branches determine replacement value; the incumbent endpoint is additionally needed for regression. Thus paired evaluation lacks exactly one record for this joint assessment. Appendix B.1–B.2 gives the constructions.

2.3 FROM LABELS TO RELEASE REQUIREMENTS

Define endpoint loss and *unearned change* by

$$\begin{aligned} L(x) &= \mathbf{1}\{r_1(h_1(x)) < r_0(h_0(x))\}, \\ J(x) &= \mathbf{1}\{q(h_1(x)) \neq q(h_0(x)),\ u_1(h_1(x)) \not\succ u_1(h_0(x))\}. \end{aligned} \tag{2}$$

For a workload $\mathcal{W}$ of size $N > 0$ and tolerances $\epsilon_L, \epsilon_J \in (0, 1]$, the basic contract requires

$$\frac{1}{N}\sum_{x\in\mathcal{W}} L(x) < \epsilon_L, \qquad \frac{1}{N}\sum_{x\in\mathcal{W}} J(x) < \epsilon_J. \tag{3}$$

Both rates use the full workload denominator. Counting ties as unearned expresses a strict-improvement requirement, not a claim that tied changes are harmful. Labels also support other predeclared clauses, such as beneficial changes outnumbering harmful ones. Paired endpoints determine L and the departure set; only replacement labels on departures require additional execution.

3 COUNTERFACTUAL REPAIR AND RISK-LIMITING RELEASE

3.1 FEEDBACK AND EXECUTABLE REVISION

The proposer receives the current program, edit interface, endpoint summaries, and beneficial/harmful/tied examples with requests, handoffs, outcomes, traces, and costs. The comparison evaluates the replacement; execution context helps the proposer infer a control rule to revise. A harmful terminal-for-write replacement can motivate a guard against stopping before a supported, requested action. A tied extra operation can motivate conditional rather than universal use. We instruct the proposer to retain mechanisms behind beneficial changes and revise harmful or tied ones, deriving control rules rather than memorizing task identities.

AgentDojo proposals replace complete `policy.py` and `prompts.py` files. Validation checks syntax, schema, and checkpoint compatibility; task IDs and hard-coded answers are prohibited. Each executable revision receives fresh endpoints and a child-on-parent census on the fixed development workload, because both its handoff and continuation may change. The evaluator, utility, projection, and contract remain fixed; invalid proposals and execution failures consume opportunities. An empty evaluated-candidate set skips selection; if no revision is evaluable within budget, the release is withheld. Failed candidates and records can seed another batch. Selection determines which program enters a separate confirmation audit (Algorithm 1); it does not itself authorize release.

For candidate A_i, let S_i, L_i, and J_i denote development success, regression, and unearned-change counts, and let Q_i indicate satisfaction of the complete development qualification rule, including positive success gain. Selection uses the lexicographic score

$$\phi_i = \big(Q_i,\ S_i - S_0,\ -J_i,\ -L_i,\ -i\big), \qquad i^\star = \arg\max_{i \in \mathcal{I}}^{\text{lex}} \phi_i, \tag{4}$$

where S_0 is incumbent development success and $\mathcal{I}$ contains candidates with complete development evaluations. This separates the information used to propose a revision from the rule used to select it. In the method comparisons, all arms use this evaluator and selector; only CounterGraft exposes the counterfactual records to its proposer. Batch-stopped search completes the first batch containing a qualified candidate before selecting; fixed-budget search evaluates every declared opportunity. If no candidate qualifies by budget exhaustion, the highest-ranked failed candidate is frozen. Confirmation permits neither revision nor runner-up substitution.

3.2 On-demand confirmation

The audit asks whether the unopened labels could still change the release decision. If every possible completion passes the contract, it certifies; if every completion fails, it withholds. Only when both verdicts remain possible does it use statistical evidence to certify early or acquire another label.

Before acquisition, the audit binds the incumbent and frozen candidate, confirmation workload, execution interface, utility, projection, and contract. Their potential records form a stable finite table. Mandatory endpoint evaluation determines regressions and the M departures. Labels are then opened in a uniform random order, independent of unrevealed labels conditional on that table. The integer budgets are $B_L = \lceil \epsilon_L N \rceil - 1$ and $B_J = \lceil \epsilon_J N \rceil - 1$; an endpoint loss count exceeding B_L immediately withholds the candidate.

To formalize the completion checks, let $\Theta_M = \{(P, H, T) \in \mathbb{Z}^3_{\geq 0} : P + H + T = M\}$ contain possible label totals, and let $\mathcal{C} \subseteq \Theta_M$ contain those satisfying all handoff clauses. The basic clause is $H + T \leq B_J$. Observed counts s_t leave feasible totals $\Theta_t = \{\theta \in \Theta_M : \theta \geq s_t\}$. Thus $\Theta_t \subseteq \mathcal{C}$ certifies, whereas $\Theta_t \cap \mathcal{C} = \varnothing$ withholds. For the basic clause, j_t observed unearned changes bound the final count K by $j_t \leq K \leq j_t + M - t$.

When completion checks are inconclusive, we compare the likelihood of the observations under compatible totals with their best fit under an incompatible total. For $\theta = (n_b, n_h, n_\tau)$, the ordered-prefix likelihood is $\ell_t(\theta) = \prod_c (n_c)_{s_{t,c}} / (M)_t$, where $(n)_s$ is a falling factorial; infeasible totals have zero likelihood. Fix mixture weights π_M on $\mathcal{C}$ before acquisition (uniform in our implementation), and define

$$E_t = \frac{\sum_{\theta \in \mathcal{C}} \pi_M(\theta) \ell_t(\theta)}{\max_{\theta \in \Theta_M \setminus \mathcal{C}} \ell_t(\theta)}. \tag{5}$$

For a declared $\alpha \in (0, 1)$, certify when $E_t \geq 1/\alpha$. Completion checks run first, including at $t = 0$, so empty feasible classes are resolved exactly. We enumerate totals in log space without approximation or truncation. Acquisition failure withholds.

3.3 Guarantees across adaptive releases

Proposition 3 (Bound-release false-certification control)**.** *Under the bound-table and uniform-sampling conditions, a contract-incompatible artifact satisfies* $\Pr_\rho(\text{CERTIFY} \mid \textit{bound table}) \leq \alpha$*, where* ρ *is the acquisition permutation.*

Algorithm 1 CounterGraft: development repair and independent release

Given: Incumbent A_0, initial candidate A, development tasks $\mathcal{W}_d$, confirmation tasks $\mathcal{W}_c$, and declared budget, contract, and risk α.

I. Repair and freeze

1 Propose and validate a batch of revisions using current feedback.
2 Evaluate each revision on $\mathcal{W}_d$; compute its score ϕ_i using Eq. (4).
3 Select from candidates with completed evaluations $\mathcal{I}$:

$$i^\star \leftarrow \arg\max_{i\in\mathcal{I}}^{\text{lex}} \phi_i, \qquad A \leftarrow A_{i^\star}.$$

4 Use A and its feedback for the next batch, until the stopping rule is met.
5 Freeze $A^\star \leftarrow A$ and bind its confirmation on $\mathcal{W}_c$.

II. Audit the frozen candidate

6 Evaluate endpoints. If $\sum_{x\in\mathcal{W}_c} L(x) > B_L$, return WITHHOLD.
7 Form compatible totals $\mathcal{C}$, fix π_M, and randomly order the M departures.
8 Set $s_0 = (0,0,0)$ and $\Theta_0 = \Theta_M$. At each prefix t, check *in order*:

$$\Theta_t \cap \mathcal{C} = \varnothing \longrightarrow \text{WITHHOLD}$$
$$\Theta_t \subseteq \mathcal{C} \longrightarrow \text{CERTIFY}$$
$$E_t \geq 1/\alpha \longrightarrow \text{CERTIFY} \quad \text{[Eq. (5)]}.$$

9 If unresolved, execute the next child-on-parent branch, obtain label z, and update:

$$s_{t+1} \leftarrow s_t + e_z, \qquad \Theta_{t+1} \leftarrow \{\theta \in \Theta_M : \theta \geq s_{t+1}\}.$$

Here $s_t = (P_t, H_t, T_t)$ and e_z increments the acquired label's count. Only certified candidates are eligible to replace the incumbent.

Deterministic checks cannot falsely certify. Under any incompatible total, E_t is dominated by the likelihood ratio of the compatible mixture to that total, a nonnegative supermartingale starting at one. Ville's inequality bounds threshold crossing under arbitrary stopping. The guarantee concerns acquisition randomness on the bound workload; it does not assume independent tasks or provide a new-workload guarantee (Appendix B.3).

Corollary 1 (Adaptive-family risk control)**.** *If each release uses a fresh conditionally uniform permutation and a history-measurable risk allocation with* $\sum_k \alpha_k \leq \alpha_{\text{family}}$ *almost surely, then* $\Pr(\textit{any false certification}) \leq \alpha_{\text{family}}$.

Conditioning on each release's history and bound table, then taking expectations and a union bound, proves the result without independence across releases. Opened development or audit records may guide later proposals; each frozen artifact requires its own confirmation.

4 Experiments

4.1 Setup and comparisons

We evaluate complete repair methods, feedback, behavioral mechanisms, and release. Baselines are adapted Self-Refine (Madaan et al., 2023), GEPA (Agrawal et al., 2026), and Hierarchical Self-Improvement (HSI; Zhou, 2026). AgentDojo (Debenedetti et al., 2024) uses GPT-5.4 to propose executable policy and prompt revisions, with GPT-4.1, GPT-5.2, or GPT-5.4 executing. Each method starts from the same three initial targets per executor and conducts one search per method–target pair. ALFWorld (Shridhar et al., 2020) uses Qwen3-8B for proposal and execution over a shared policy-module pool. Adaptations preserve self-critique, reflective evolution, or hierarchical harness revision while standardizing the editable interface and evaluation protocol (Appendix H).

AgentDojo permits five batches of eight proposals on 64 development tasks. A common evaluator computes endpoints and counterfactual labels for qualification and final selection; baseline proposers receive ordinary outcomes and traces, not child-on-parent records or labels. After completing

a batch, search stops if any candidate qualifies. The common selector follows Eq. (4), including selection of the best failed candidate at budget exhaustion. One program per method–target pair is frozen before confirmation, without substitution or revision afterward. Each target has 159 confirmation tasks; the three targets give 477 target–task evaluations per executor. ALFWorld runs all 12 opportunities and freezes one selection before 134-task confirmation.

Utility is lexicographic $[r, -c_{\text{model}}, -c_{\text{tool}}]$: task adequacy precedes model turns and tool calls. AgentDojo requires positive development gain and strict 10% regression and unearned-change budgets; confirmation allows at most 15 of each per target. The 134-task ALFWorld contract requires $L \leq 6$, $H + T \leq 6$, $P > H$, $P + H > 0$, and improved endpoint success. The method comparison tests distinct proposal mechanisms under common selection; the fixed-budget F2 study changes only proposer-visible feedback. The GPT-4.1 CounterGraft programs are also analyzed against their initial successors to explain the repairs.

Table 1: **Task performance and certified successors across models.** Results are from our adapted implementations, not quoted scores. Success is task success (%); Certified counts admitted artifacts. Each AgentDojo executor has three 159-task targets; ALFWorld evaluates one selection on 134 tasks. Proposal caps are 40 per AgentDojo target and 12 for ALFWorld, with common evaluation and selection.

	AgentDojo						ALFWorld
	GPT-4.1		GPT-5.2		GPT-5.4		Qwen3-8B
Method	Success ↑	Cert.	Success ↑	Cert.	Success ↑	Cert.	Success ↑
Parent	62.9	—	71.9	—	80.1	—	70.9
Adapted Self-Refine	91.8	0/3	88.7	1/3	91.2	1/3	72.4
Adapted GEPA	93.7	1/3	92.2	1/3	93.7	2/3	73.9
Adapted HSI	94.5	1/3	93.5	2/3	94.8	2/3	74.6
CounterGraft	**95.8**	**2/3**	**95.4**	**3/3**	**96.2**	**3/3**	**75.4**
Δ vs. parent (pp)	+32.9	—	+23.5	—	+16.1	—	+4.5
Δ vs. best baseline (pp)	+1.3	+1	+1.9	+1	+1.5	+1	+0.7

4.2 Higher task success with better behavioral replacements

CounterGraft improves both task success and release eligibility across the tested executors (Table 1). On AgentDojo, success reaches 95.8%, 95.4%, and 96.2% under GPT-4.1, GPT-5.2, and GPT-5.4, respectively, exceeding adapted HSI by 1.3, 1.9, and 1.5 percentage points. The same ordering holds on ALFWorld, where CounterGraft solves 101/134 tasks. Across the nine AgentDojo selections, CounterGraft certifies eight, versus two for Self-Refine, four for GEPA, and five for HSI.

The higher success accompanies better behavioral replacements (Table 2). Under GPT-4.1, CounterGraft reduces harmful replacements from HSI's 22 to 18 and non-improving replacements from 43 to 33. On ALFWorld, its selected program makes no harmful or tied replacements; HSI makes one and six. To examine the code improvements behind these scores, we compare the same selected programs with their initial, unrepaired successors: across GPT-4.1 targets F2/F4, success rises from 297 to 308, harmful replacements fall from 23 to seven, and regressions fall from 15 to five. Both repaired programs pass the contract; the third target, F1, remains ineligible and is withheld (Figure 3). Harmful counts measure utility-degrading replacements directly; release decisions additionally enforce the declared budget on ties (Appendix F.3).

Improving which actions the successor takes. F4 makes 65 departures both before and after repair, while $P/H/T$ changes from $43/14/8$ to $58/5/2$. Across F2/F4, 73 departures turn incumbent failures into repaired successes. Writing adequacies as $(r_0(h_0), r_1(h_0), r_1(h_1))$, 68 are $(0, 0, 1)$: the repaired continuation still fails from the incumbent handoff and succeeds only from its own. The other five are $(0, 1, 1)$, where the repaired policy succeeds from either handoff. This decomposition locates the recoveries in the choices made by the program, beyond its ability to continue from the incumbent state. Conversely, repair eliminates all eight observed $(1, 1, 0)$ failures, in which the successor succeeds from the incumbent handoff but fails from its own. Harmful write-to-terminal

Table 2: **Behavioral quality of the selected programs.** L counts endpoint regressions; $P/H/T$ are beneficial/harmful/tied replacements; $M = P + H + T$ and $J = H + T$. AgentDojo aggregates three targets, each audited separately with budgets $L, J \leq 15$. ALFWorld uses $L, J \leq 6$, $P > H$, $P + H > 0$, and improved endpoint success.

Method	Success ↑	$L \downarrow$	$P \uparrow$	$H \downarrow$	T	M	$J \downarrow$	Release
A. AgentDojo / GPT-4.1: three held-out targets								
Parent	300/477	—	—	—	—	—	—	Reference
Adapted Self-Refine	438/477	18	139	35	37	211	72	0/3 certified
Adapted GEPA	447/477	14	149	24	28	201	52	1/3 certified
Adapted HSI	451/477	13	156	22	21	199	43	1/3 certified
CounterGraft	**457/477**	**12**	**163**	**18**	**15**	196	**33**	**2/3 certified**
B. ALFWorld / Qwen3-8B: frozen selection								
Parent	95/134	—	—	—	—	—	—	Reference
Adapted Self-Refine	97/134	3	4	2	8	14	10	Withhold
Adapted GEPA	99/134	2	5	1	7	13	8	Withhold
Adapted HSI	100/134	2	6	1	6	13	7	Withhold
CounterGraft	**101/134**	**0**	6	**0**	**0**	6	**0**	**Certify**

replacements fall from eleven to zero, while beneficial cancellations remain. Cost-reducing departures at unchanged adequacy increase from 29 to 43; the latter save 366 model turns relative to their child-on-parent branches.

On a held-out banking task, original F4 computes a four-unit refund but terminates without paying. It succeeds from the incumbent's post-refund handoff. Development feedback contains an analogous file-deletion failure; the generated repair tracks requested side effects and completed writes, prevents premature stopping, and grounds write arguments in task evidence. It executes the held-out refund successfully (Figure 1; code and records in Appendix A.3).

4.3 Isolating the feedback contribution

Counterfactual feedback improves free-form repair. On the same initial F2 program, CounterGraft produces 48 beneficial and three harmful replacements, compared with 37 and ten under endpoint-and-trace feedback, while solving 157 versus 156 tasks (Figure 2a). Departure counts are nearly unchanged at 57 versus 58. This experiment fixes the proposer, executor, editable files, 64-task development workload, and 16 proposal opportunities. Both arms compute development counterfactual labels for the same full-information selector. The baseline proposer receives endpoints, selected actions, complete traces, tool returns, and costs; CounterGraft additionally exposes child-on-parent outcomes and labels. Thus both arms can select on behavioral quality; the treatment supplies that information to the code-revision process itself. Through the release decision, the baseline uses 27.81M tokens and CounterGraft uses 28.67M, a 3.1% increase (Appendix G.1).

The same information changes module selection. On 200 ALFWorld tasks (Figure 2b), holding module-level endpoint feedback, edit space, and execution fixed, child-on-parent labels change selection from Heat + Pick-two to Heat. Success rises from 152 to 153 (incumbent: 151), and $P/H/T$ changes from $2/3/20$ to $2/0/0$. Endpoint regressions fall from two to zero; only the counterfactual-guided program passes the strict 5% unearned budget and directional clauses $P > H$, $P + H > 0$. This fixed-module study complements free-form revision; Appendix D reports a separate localized-feedback study.

4.4 Release decisions and acquisition cost

The audit reaches release decisions without opening every counterfactual branch. In matched F2, it certifies CounterGraft after 34 of 57 replays at $\alpha = 0.025$, while the endpoint-and-trace candidate is withheld after 44 of 58. Certification occurs while completion bounds still allow either verdict; withholding occurs when observed violations exceed the budget. Post-decision censuses verify both outcomes (Appendix G).

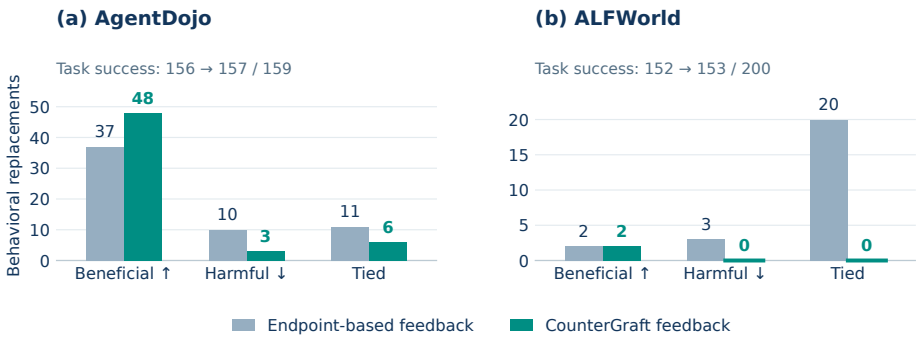


Figure 2: **Matched feedback improves behavioral quality under two executors.** (a) AgentDojo F2 uses the same proposer, executor, edit space, selector, and 16 opportunities per arm. (b) The ALFWorld 200-task study fixes module-level endpoint feedback and edit space. Both studies test the effect of exposing child-on-parent evidence. Paired bars show beneficial, harmful, and tied replacements; task success is reported above each panel.

Sequential evidence provides savings beyond those available from completion checks alone. On the development repair F3 and original-panel F2/F4 repairs, it saves 7–25 continuations at $\alpha = (0.025, 1/120, 1/240)$, holding the program, contract, record table, and acquisition order fixed (Figure 4). Across 200 predeclared orders of a fixed SciFact table, mean acquisition is 16.44, versus 20 for completion checks and 26 for census; 179 orders stop statistically at $\alpha = 0.025$. Appendix E reports the contracts and full acquisition counts.

Even without a statistical stop, completion checks avoid unnecessary replay. They resolve all 13 original prospective AgentDojo audits; in the same-backbone cohort, opening 66 of 101 records saves 34.7% of continuations and 41.2% of replay tokens. These savings concern audit acquisition, excluding development, mandatory endpoints, and verification census.

The additional behavioral requirement also changes which updates are admitted. On 16 frozen scenarios, an endpoint negative-flip gate accepts four contract-incompatible updates, while strict action agreement rejects all five compatible ones. Applying PACE and the SGM product rule to our 17 frozen releases yields agreement with the complete contract on 9 and 12 decisions (Appendix F). These comparisons evaluate admission under our contract; the methods retain their native endpoint guarantees.

4.5 Certified successors enter subsequent execution

Certified programs are actually installed in both prospective cohorts (Table 18). Each cohort installs one successor and retains the incumbent after every other audit. In the fixed-GPT-4.1 cohort, all five candidates improve task success by 23.6–39.3 percentage points, but only one satisfies the complete contract. One lineage progresses from two rejected candidates to a certified fourth-generation program. The same-backbone GPT-5.4 cohort produces 22 modified candidates and eight audits from 36 opportunities. These runs connect the release verdict to which program executes next.

Rejection supplies feedback for a new repair. F3 shows how improvement can continue after a failed release. Its first repair retains a 20-task development gain, but confirmation finds 21 unearned changes and withholds it. Those records inform a new batch with six of eight qualified candidates. The newly frozen selection solves 151/161 tasks versus its incumbent's 91, with five regressions and $58/1/3$ beneficial/harmful/tied labels. Its own audit certifies after 24/62 continuations. Certification therefore follows a new program and separate confirmation evidence, while the earlier rejection remains in force (Appendix C).

The comparison also applies to updates beyond generated repairs. A panel of 17 frozen updates covers 9,291 evaluation units across evidence QA, code generation, AppWorld, retrieval routing, and stateful agents, spanning six GPT/Qwen backbones and a cached bank of 15 RAG models (Table 13). The two release clauses detect complementary failures: each NQ router passes endpoints

but makes one beneficial and 13 harmful departures, whereas the FEVER producer makes 80 beneficial departures yet exceeds its regression budget. Appendix F reports the component configurations, replay-stability checks, and agreement with census across 5,600 empirical/synthetic acquisition orders.

5 Related Work

Agent self-improvement. Self-Refine uses iterative self-critique, and Reflexion carries verbal feedback into later trials (Madaan et al., 2023; Shinn et al., 2023). TextGrad and GEPA optimize compound systems through textual feedback and trajectory reflection (Yuksekgonul et al., 2025; Agrawal et al., 2026). ADAS, the Darwin Gödel Machine, and HSI extend search to workflows, executable agents, and hierarchical harnesses (Hu et al., 2025; Zhang et al., 2025; Zhou, 2026). CounterGraft adds an executed incumbent-relative alternative: it evaluates which replacements help under the revised policy and uses that evidence for revision and release.

Counterfactual execution and attribution. Causal Agent Replay intervenes on trajectory steps and re-executes under a fixed stochastic policy to estimate step effects (Shah, 2026). CausalFlow localizes failure-inducing steps and constructs counterfactual repairs (Bonagiri et al., 2026). Our contrast fixes the candidate policy and compares its selected handoff with the incumbent-selected handoff. This contrast captures both task and cost effects, supplying revision feedback and separately bound release evidence.

Successor evaluation and promotion. High-confidence policy improvement and SPIBB impose incumbent-relative performance requirements (Thomas et al., 2015; Laroche et al., 2019). PACE, SGM, and SEA gate updates through paired tests, cross-update risk allocation, or statistical certificates (Shawn, 2026; Wu et al., 2025; Sengupta, 2026). CounterGraft combines endpoint-regression limits with replacement-value requirements; our fixed-record comparisons quantify the resulting differences in admission.

Sequential certification. Confidence sequences support time-uniform inference (Howard et al., 2021), including sampling without replacement (Waudby-Smith & Ramdas, 2020); risk-limiting audits use sequential evidence to bound erroneous decisions while avoiding a full census (Stark, 2008; Shekhar et al., 2023). CounterGraft translates behavioral release requirements into constraints on finite label totals. Completion checks and mixture evidence support on-demand certification, with error spending across adaptive releases.

6 Discussion and Conclusion

CounterGraft turns an incumbent-relative execution comparison into feedback for revising agent programs and evidence for releasing them. The experiments show higher task success across executors and improved behavioral quality under matched feedback. Repairs recover requested actions, retain useful cancellations, and reduce execution cost. At an unchanged departure count, they improve which changes the successor makes.

The comparison requires recoverable handoffs and a declared utility; independent confirmation controls false certification on the bound workload. Different contracts can require strict improvement or tolerate equivalent replacements. Prospective installations demonstrate the operational loop: rejected candidates inform further repair, while certified successors become incumbents. Longer lineages can test how gains and audit costs evolve. CounterGraft makes the value of a behavioral replacement an explicit criterion for both constructing successors and deciding which ones inherit execution.

AI USE STATEMENT

LLMs are central to this work: as the agents under evaluation and as the proposers that revise executable policies and prompts or select policy modules (Sections 3–4). Proposed changes are evaluated through execution and a separate release audit. In preparing this manuscript, AI tools assisted with literature search, text revision and language polishing, and figure and LaTeX formatting.

REPRODUCIBILITY STATEMENT

Section 2 defines the comparison and contract, and Algorithm 1 describes repair and confirmation. Appendix A specifies execution semantics and behavioral projections; Appendices B.1–B.3 provide proofs. Appendices C, D, and G document repair protocols, selection rules, reported configurations, and outcomes. The supplement includes machine-readable reported counts and a reference implementation of the finite-population evidence calculation; these do not replace the original execution harness or raw run logs.

ETHICS STATEMENT

The tool-action experiments use controlled benchmark environments and restorable execution states. A successful audit establishes compatibility with a declared workload and utility, not unrestricted safety or permission to perform real-world side effects. The release interface should be used only where state restoration and action accounting faithfully represent the declared comparison.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf.

Akash Bonagiri, Devang Borkar, Gerard Janno Anderias, Setareh Rafatirad, and Houman Homayoun. Causalflow: Causal attribution and counterfactual repair for llm agent failures. *arXiv preprint arXiv:2605.25338*, 2026.

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 A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 82895–82920. Curran Associates, Inc., 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/97091a5177d8dc64b1da8bf3e1f6fb54-Paper-Datasets_and_Benchmarks_Track.pdf.

Steven R Howard, Aaditya Ramdas, Jon McAuliffe, and Jasjeet Sekhon. Time-uniform, nonparametric, nonasymptotic confidence sequences. *The Annals of Statistics*, 49(2):1055–1080, 2021.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International conference on machine learning*, pp. 3652–3661. PMLR, 2019.

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. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Biswa Sengupta. Self-evolving agents with anytime-valid certificates. *arXiv preprint arXiv:2607.00871*, 2026.

Jaineet Shah. Causal agent replay: Counterfactual attribution for llm-agent failures. *arXiv preprint arXiv:2606.08275*, 2026.

Zayx Shawn. Pace: Anytime-valid acceptance tests for self-evolving agents. *arXiv preprint arXiv:2606.08106*, 2026.

Shubhanshu Shekhar, Ziyu Xu, Zachary Lipton, Pierre Liang, and Aaditya Ramdas. Risk-limiting financial audits via weighted sampling without replacement. In *Uncertainty in Artificial Intelligence*, pp. 1932–1941. PMLR, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Philip B Stark. Conservative statistical post-election audits. 2008.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pp. 2380–2388. PMLR, 2015.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 809–819, 2018.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 7534–7550, 2020.

Ian Waudby-Smith and Aaditya Ramdas. Confidence sequences for sampling without replacement. *Advances in neural information processing systems*, 33:20204–20214, 2020.

Xuening Wu, Shenqin Yin, Yanlan Kang, Xinhang Zhang, Qianya Xu, Zeping Chen, and Wenqiang Zhang. Sgm: A statistical godel machine for risk-controlled recursive self-modification. *arXiv preprint arXiv:2510.10232*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents, 2025a. *URL https://arxiv. org/abs/2505.22954*, 2025.

Jiarui Zhang, Xiangyu Liu, Yong Hu, Chaoyue Niu, Fan Wu, and Guihai Chen. Ragrouter: Learning to route queries to multiple retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 38:15709–15737, 2026.

Tailin Zhou. Hierarchical self-improvement: A framework for task-specific evolvable agent harnesses. *arXiv preprint arXiv:2608.08466*, 2026.

A Execution and Feedback Interfaces

A.1 Bound handoffs and branch execution

A handoff contains the boundary choice and the state required to resume execution. The execution interface binds the interpretation of this boundary before comparison: the action is either pending and applied once after restoration, or already applied and not repeated. The same convention governs both candidate branches. Their utility comparison holds the candidate policy fixed, not its realized trajectory: different retained states can induce different subsequent actions. When histories differ, the estimand concerns the complete handoff, including its state, rather than an isolated action at an otherwise identical state.

In AgentDojo, the protected carrier stores the complete post-action world, dialogue prefix, interaction log, policy state, and next turn. Child-on-parent execution restores a deep copy of the carrier produced after the incumbent-selected action and resumes with candidate policy code. The boundary action is not executed again. Its effect and cost enter once under the carrier accounting convention. Branches are evaluated in the controlled benchmark environment; restoring a post-refund carrier does not mean issuing a second payment. Applying this interface to an external service requires an equivalent isolated or faithfully restorable execution environment.

The projection q determines whether the retained choices count as different behaviors, whereas the carrier determines what is actually executed. They serve different purposes: a compact behavior signature need not reproduce the execution state. For example, the F4 comparison distinguishes a terminal choice from the incumbent's write action; restoration still includes the complete post-action carrier. Labels depend jointly on the declared projection, utility, and continuation. Changing any of these defines a different evaluation object.

Behavioral projections. For AgentDojo, $q(h)$ is the canonical selected tool-call block. Each call contains the tool name and schema-normalized arguments. JSON key order is ignored and numeric values are normalized by the schema (e.g., 5 and 5.0 agree). String contents, entity identifiers, amounts, dates, and recipients must match. Semantically ordered lists retain their order; parallel, semantically independent calls within one assistant turn are placed in a canonical order. Terminal `STOP`, text-only answers, and tool calls are distinct actions; a terminal action is not a missing record. Two blocks agree only if their complete canonical representations agree. Thus, `get_most_recent_transactions(n=20)` and the same call with `n=100` differ, as do a refund and termination, whereas a change in JSON key order alone does not create a departure. Message histories, tool returns, and execution state remain in the carrier but do not create a departure merely because their bytes differ.

In ALFWorld, $q(h)$ is the canonical primitive environment action: its verb, object identity or type, target receptacle or location, and required parameters, with `STOP` treated separately. Module names such as Heat and Pick-two are attribution metadata; a departure requires a change in the action produced at the declared boundary. In QA controllers, the projection records the controller route, answer mode, and selected evidence identifiers. Retrieval interfaces record the backend or index, canonical query, top-k, and required filters. Producer/verifier interfaces record the selected branch and its bound input. Prompt/policy agents compare the canonical action at the declared execution boundary. These projections specify behavioral equality without equating it with carrier equality.

Table 3: Execution objects underlying the comparison interface.

Setting	Handoff and outcome interpretation
Agent environments	Action or checkpoint with restorable environment and policy state; the candidate resumes under the bound action convention.
Evidence QA	Producer or verifier choice with the information supplied to the downstream controller.
Code generation	Draft, repair, or abstention mode within the generation controller; generated code is tested by EvalPlus.
Retrieval routing	A selected route with its corresponding cached prediction in the fixed public output bank. No new live model execution is implied.

A.2 Utility and interpretation of ties

AgentDojo repair compares $u = [r, -c_{\text{model}}, -c_{\text{tool}}]$ lexicographically. Task adequacy has priority; among equally adequate executions, fewer model turns are preferred, followed by fewer tool calls. A cheaper failure cannot outrank a successful execution. A tie denotes equivalence under this declared ordering, not universal equivalence or absence of every possible benefit.

The basic contract budgets $H + T$, representing a preference for changes that demonstrate strict utility improvement. This is a release requirement, not a claim that every tied implementation is intrinsically undesirable. The separate P, H, T counts permit contracts that treat ties differently. For strict rate tolerance ϵ on N tasks, the maximum integer violation count is $\lceil \epsilon N \rceil - 1$; at $N = 159$ and $\epsilon = 0.1$, this is 15.

A.3 Feedback and generated control rules

The free-form proposer receives the current `policy.py` and `prompts.py`, the editable interface, and a bounded JSON feedback object. Feedback includes workload-level endpoint summaries, selected failures and regressions, a counterfactual summary, and examples grouped as beneficial, harmful, or tied. Each example records the request, incumbent and candidate choices, endpoint and child-on-parent outcomes, and the label. Execution traces and costs provide context for turning a label into a program change. The repair instruction is:

> Preserve mechanisms behind beneficial replacements; repair mechanisms behind harmful replacements; avoid behavior churn that ties under the declared adequacy-then-cost ordering. Do not memorize examples.

The proposer returns complete replacement strings for the two permitted files. Task identifiers and hard-coded answers are prohibited. Validation checks the output schema, Python syntax, and checkpoint interface; validation failure is retained as an opportunity outcome.

Development evidence. The final F4 development packet includes a request to list shared file identifiers and delete the largest file. The incumbent retains `delete_file(file_id="11")`, while the candidate terminates. Candidate adequacy is zero under its terminal handoff and one under the incumbent handoff, with respective utilities $[0, -2, -1]$ and $[1, -3, -2]$. The departure is harmful. The packet summary reports 22 beneficial, seven harmful, and five tied departures. This file-deletion feedback is distinct from the held-out banking task below.

Executable revision. The original stopping condition accepts any nonempty text:

```
def should_stop(self, content, state, turn):
    return bool(content.strip())
```

The repair identifies requests involving side effects and tracks executed read and write calls. The following excerpt blocks termination in a relevant state:

```
if (
    profile.get("side_effect")
    and int(state.get("write_calls", 0)) == 0
    and not _NONANSWER_RE.search(text)
):
    return False
```

The accompanying prompt requires completion of supported requested actions after the necessary lookup or computation, while grounding every write argument in the request or task-relevant evidence. This excerpt is part of a task-conditioned controller, not a complete specification of all its stopping behavior.

Held-out refund task. A dinner costs 12 units; the friend transfers ten although their share is six. The user requests checking the transfer and refunding the excess. Original F4 computes the four-unit refund but terminates without issuing it. Restoring the incumbent's post-refund carrier allows the

same original successor to confirm completion. The repaired program retrieves the transaction and executes `send_money(amount=4, ...)`. The original terminal replacement is harmful. The

Table 4: F4 refund case under the declared adequacy-then-cost ordering. The banking task is absent from development feedback.

Execution	Adequacy	Utility
Incumbent endpoint	1	$[1, -3, -2]$
Original successor endpoint	0	$[0, -2, -1]$
Original successor on incumbent carrier	1	$[1, -3, -2]$
Repaired successor endpoint	1	$[1, -3, -2]$
Repaired successor on incumbent carrier	1	$[1, -15, -2]$

repaired own handoff is beneficial under the declared cost ordering: both repaired branches succeed, but the restored branch incurs repeated confirmation and 15 model turns. This cost comparison is relative to that restored branch, not a claim that repair saves 12 turns over the original endpoint. The example links development feedback from file deletion to a control rule that also completes the held-out refund.

B Identification and Certification Proofs

B.1 Endpoint non-identification

Proof of Proposition 1. Fix an executable departure with $q(h_1) \neq q(h_0)$ and scalar utility in $[0, 1]$. Construct two admissible tables with identical handoffs, endpoint adequacy scores, and all cells except $Y_1(h_0)$. In both, set $u_0(h_0) = 0.5$ and $u_1(h_1) = 0.6$. Set $u_1(h_0) = 0.4$ in the first and 0.8 in the second, keeping its adequacy fixed. The same endpoint record therefore corresponds to a beneficial replacement in the first table and a harmful one in the second. Any rule based only on that record gives the same answer in both worlds and cannot identify the label uniformly. Repeating an already observed endpoint supplies no information about the unconstrained missing cell. The construction is a witness within the nonparametric class; it does not assert ambiguity for every particular task or a model with additional structural constraints. □

B.2 Sufficiency and primitive-query minimality

Proof of Proposition 2. The two endpoints determine L, and the two candidate-continuation outcomes determine the utility comparison. With the selected handoffs and q given, the three records are sufficient.

For necessity, consider executable departures and a binary adequacy space $\{0, 1\}$. Removing $Y_1(h_0)$ leaves the indistinguishable pair just constructed. Removing $Y_0(h_0)$, fix candidate adequacy $r_1(h_1) = 0$ and vary the missing incumbent adequacy from zero to one; all retained records stay fixed while L changes from zero to one. Removing $Y_1(h_1)$, fix incumbent adequacy $r_0(h_0) = 1$ and vary the missing candidate adequacy from one to zero, again changing L. Thus, omitting any one record prevents uniform joint identification. Every smaller subset omits at least one such record, establishing the claim. The result concerns primitive payoff queries, not an oracle returning the label directly, and allows degenerate instances to need fewer queries. □

B.3 Single-release risk control

Proof of Proposition 3. Condition on the full bound record table. An endpoint violation yields deterministic withholding. Otherwise, for a contract-incompatible artifact let $\theta_0 \notin \mathcal{C}$ be its true departure-label total. At every observed prefix the true total remains feasible, so $\Theta_t \subseteq \mathcal{C}$ cannot hold. Hence deterministic completion cannot falsely certify.

Write $Z_{1:t}$ for the label prefix and let

$$Q_t(Z_{1:t}) = \sum_{\theta \in \mathcal{C}} \pi_M(\theta) \ell_t(\theta), \qquad R_t^{\theta_0} = \frac{Q_t(Z_{1:t})}{\ell_t(\theta_0)}$$

on prefixes with positive probability under θ_0. When $\mathcal{C}$ is nonempty, $R_0^{\theta_0} = 1$. If $\mathcal{S}_t$ is the support of the next label under the true total, then

$$\mathbb{E}_{\theta_0}[R_{t+1}^{\theta_0} \mid Z_{1:t}] = \frac{\sum_{z \in \mathcal{S}_t} Q_{t+1}(Z_{1:t}, z)}{\ell_t(\theta_0)} \leq \frac{Q_t(Z_{1:t})}{\ell_t(\theta_0)}.$$

The inequality permits the compatible mixture to assign mass outside the next-step support of the true total. Thus $R_t^{\theta_0}$ is a nonnegative supermartingale. Whenever the statistical boundary is evaluated, its denominator includes $\ell_t(\theta_0)$, so $E_t \leq R_t^{\theta_0}$. Ville's inequality gives

$$\Pr_{\theta_0}\{\exists t : E_t \geq 1/\alpha\} \leq \Pr_{\theta_0}\{\sup_t R_t^{\theta_0} \geq 1/\alpha\} \leq \alpha.$$

Empty compatible sets and prefixes with no compatible completion are withheld by the preceding deterministic checks. A full census determines the exact contract verdict. These observations cover support boundaries without division by a zero likelihood. Acquisition failure results in withholding and cannot increase false certification. □

The evidence statistic need not itself be a supermartingale: domination by the true-null likelihood-ratio process suffices. The probability is over acquisition order conditional on the table, not over new deployment tasks or repeated model executions. No independence among workload units is required.

B.4 Adaptive release families

Proof of Corollary 1. Let $\mathcal{H}_k^-$ be the history before opportunity k, $\mathcal{B}_k$ its bound execution table, and F_k the event of false certification. Condition on $(\mathcal{H}_k^-, \mathcal{B}_k)$ for each opportunity. The chosen artifact, audit rules, bound table, and history-measurable α_k are fixed under this conditioning, and the acquisition order remains fresh and uniform. The single-release result gives $\Pr(F_k \mid \mathcal{H}_k^-, \mathcal{B}_k) \leq \alpha_k$. For any finite n, the union bound and iterated expectation yield

$$\Pr\left(\bigcup_{k=1}^{n} F_k\right) \leq \sum_{k=1}^{n} \mathbb{E}[\alpha_k] \leq \mathbb{E}\left[\sum_{k \geq 1} \alpha_k\right] \leq \alpha_{\text{family}}.$$

Continuity from below proves the countable-family claim. Dependence of future candidates on earlier records is allowed; conditional validity of each acquisition permutation and the almost-sure total budget constraint supply the guarantee. □

C Repair Protocol, Complete Outcomes, and Inheritance

C.1 Development and frozen confirmation

GPT-5.4 proposes complete policy and prompt replacements, and GPT-4.1 executes the incumbent and candidates. F3 is used to develop the procedure; F1, F2, and F4 are the three remaining targets. Each target uses a fixed 64-task development workload and at most five batches of eight proposal opportunities. A candidate qualifies by improving task success and keeping both endpoint regressions and unearned changes strictly below 10%. With 64 tasks, either count must therefore be at most six. Among qualified candidates, selection maximizes success gain, then minimizes unearned changes, endpoint regressions, and opportunity index. If no candidate qualifies, the highest-ranked evaluated artifact and its records inform the next batch. Generation, validation, and execution failures remain in the opportunity count.

The original repair panel uses 32, 16, and 40 opportunities for F1, F2, and F4, respectively. Its selected artifacts are frozen jointly before the shared 159-task confirmation workload is opened. Confirmation does not select a replacement candidate or edit the frozen artifact. A later release opportunity may use opened evidence, but must bind and audit its own artifact. The new matched F2 study in Appendix G uses a separate fixed two-batch protocol and must not be combined with the original F2 run.

F1 remains ineligible with 18 unearned changes. F2 and F4 pass with eight and seven, respectively. Across the two successful repairs, P rises from 89 to 111, H falls from 23 to seven, and T falls from

Table 5: **Each experiment tests a distinct part of the method.** Proposers generate code or choose modules; executors run the resulting agents. Frozen update analyses need no separate proposer.

Evidence	Model configuration	Controlled comparison or outcome
Method comparison	GPT-5.4 proposer; GPT-4.1/5.2/5.4 executors; Qwen3-8B in ALFWorld	Adapted Self-Refine, GEPA, HSI, and CounterGraft under common evaluation and selection.
Free-form feedback	GPT-5.4 proposer; GPT-4.1 executor	Matched F2 endpoint-and-trace versus counterfactual feedback; original-to-repair mechanism analysis.
Module selection	Qwen3-8B proposer and executor	Matched module-level feedback; separate localized-feedback proposal yield.
Prospective release	Fixed GPT-4.1; same-backbone GPT-5.4	Frozen candidates, independent audits, and actual successor installation.
Interface transport	Six GPT/Qwen backbones; cached RAG bank	17 frozen component, model, and routing updates over 9,291 units.

Table 6: Original repair panel, with all held-out targets retained. L is measured against the corresponding incumbent.

Target	Artifact	Success	L	P	H	T
F1	Incumbent	99	–	–	–	–
	Original	150	4	46	7	13
	Repair	149	7	52	11	7
F2	Incumbent	101	–	–	–	–
	Original	148	8	46	9	12
	Repair	154	3	53	2	6
F4	Incumbent	100	–	–	–	–
	Original	149	7	43	14	8
	Repair	154	2	58	5	2

20 to eight. Separate frozen-artifact executions on a shared workload support these comparisons; task-level counts are not independent repetitions of the repair search.

C.2 Behavioral mechanism

Across F2/F4, 73 departure records have incumbent failure and repaired success. In the order (incumbent, child-on-parent, repaired endpoint), 68 are $(0, 0, 1)$ and five are $(0, 1, 1)$. Thus 68 recoveries require the repaired handoff among the two compared alternatives. Original $(1, 1, 0)$ failures decrease from eight to zero. Harmful write-to-terminal replacements fall from eleven to zero, while useful cancellations remain. Cost-reducing beneficial departures at unchanged adequacy rise from 29 to 43; the latter save 366 model turns relative to their corresponding repaired-policy child-on-parent branches.

C.3 F3 across two release opportunities

The first repair preserves a 20-task development gain while reducing unearned changes from ten to four. Independent confirmation on 161 tasks finds 21 unearned changes and withholds it. These opened records inform a new development cycle, producing six qualified candidates in eight opportunities. The selected artifact is frozen for its own confirmation, where it solves 151 tasks versus 91 for the incumbent, with 65 endpoint wins and five losses. Its 62 departures comprise 58 beneficial, one harmful, and three tied replacements. At $\alpha = 0.025$, the audit certifies after 24 records, with reported evidence value 56.06; the later census verifies the contract. These records establish certification after further repair, not by themselves actual installation of F3.

C.4 PROSPECTIVE INSTALLATION RECORDS

In the fixed-executor cohort, five audited successors improve task success by 23.6–39.3 percentage points and pass the endpoint clause. Four fail the handoff contract. In the illustrated lineage, two withheld candidates precede `l3-g4`; on its 54-unit workload, its 22 departures comprise $18/1/3$, giving four unearned changes. It is certified and materialized as the next executor.

The same-backbone cohort uses GPT-5.4 for proposal and execution. Twelve lineages with three generation slots each supply 36 opportunities, 22 modified candidates, and eight audited artifacts. Six improve sealed task success. One passes the complete contract and is installed; seven leave the incumbent in place. All eight audit decisions agree with their later censuses. Together the cohorts demonstrate two actual installations across 13 audits, distinct from a long chain of repeatedly installed successors.

D CONTROLLED ALFWORLD FEEDBACK STUDIES

D.1 MATCHED MODULE-LEVEL FEEDBACK ON 200 TASKS

The matched study fixes the Qwen3-8B backbone, two-module edit space, module-level endpoint feedback, proposal constraints, compiler, and evaluation protocol. The added information is the child-on-parent comparison. The incumbent solves 151 tasks. Endpoint-guided selection chooses Heat + Pick-two, solves 152, and yields $2/3/20$ beneficial/harmful/tied departures. Counterfactual-guided selection chooses Heat, solves 153, and yields $2/0/0$. Endpoint regressions are two and zero, respectively. The contract applies strict 5% regression and unearned-change budgets (at most nine each), together with $P > H$ and $P + H > 0$. Only the counterfactual-guided program satisfies all clauses.

Table 7: **Added counterfactual feedback changes the selected program.** The incumbent solves 151/200 tasks. Both conditions use the same module-level endpoint feedback and edit space. Strict 5% regression and unearned-change budgets permit at most nine violations each; the directional clause requires $P > H$ and $P + H > 0$. Both programs pass the endpoint clause.

Feedback	Selected modules	Success	$P/H/T$	Decision
Module-level endpoints	Heat + Pick-two	$152/200$	$2/3/20$	Withhold
+ Child-on-parent	Heat	$153/200$	$2/0/0$	**Certify**

D.2 LOCALIZED FEEDBACK AND PROPOSAL YIELD ON 134 TASKS

A separate study uses `Qwen/Qwen3-8B` for proposal and execution. Proposal decoding uses temperature 0.8, top-$p = 0.9$, and a maximum of 400 output tokens. The three arms each receive 12 opportunities with paired seeds 150000–150011. Execution uses temperature zero, a two-shot ReAct prompt, and a 49-step interaction limit on 134 `valid_unseen` tasks. Tools, output schema, and the two-module edit space are fixed. Each proposal returns a nonempty subset of {`heat`,`pick_two`} and a rationale, which is compiled into an executable policy. The feedback fields differ as follows:

Table 8: Feedback fields in the three-arm proposal-yield study.

Condition	Proposer-visible information
Endpoint	Workload size, parent and child success, aggregate endpoint gain.
Regression-augmented	Endpoint fields plus paired wins and losses.
Counterfactual-guided	Regression fields plus per-task-type migration counts and beneficial/harmful/tied labels for each module.

The complete contract requires $L \leq 6$, $H + T \leq 6$, $P > H$, and $P + H > 0$, together with strictly higher endpoint success than the incumbent. A one-sided exact paired endpoint p-value is reported separately and is not part of ordinary qualified-opportunity yield. The predesignated

flagship opportunity additionally requires $p < 0.05$. The two arm-level success-yield comparisons use a Bonferroni-adjusted level of 0.025. These search-level comparisons remain distinct from the paired endpoint test for a selected program.

This treatment adds both counterfactual outcomes and their localization. The 200-task study above fixes module-level endpoint granularity; the two studies therefore answer related but different questions. The incumbent solves 95 tasks. The endpoint and regression-augmented selected programs

Table 9: Complete selected-program outcomes and proposal yield in the 134-task study.

Feedback	Success	Wins/losses	$P/H/T$	Unearned	Qualified opportunities
Endpoint	$100/134$	$7/2$	$7/1/11$	12	$0/12$
Regression-augmented	$100/134$	$7/2$	$7/1/11$	12	$0/12$
Counterfactual-guided	$101/134$	$6/0$	$6/0/0$	0	$8/12$

fail the contract; the counterfactual-guided program passes. The 36 proposal calls map to 18 unique executable policies in the report's execution-equivalence grouping. Collapsing equivalent counterfactual pairs gives a sensitivity comparison of four qualified outcomes out of six versus zero out of six. Opportunity yield describes generated selections, not 36 independent implementations. The separately predesignated successor's six endpoint wins and zero losses are also distinct from the search-level yield count.

E Acquisition and Release-Rule Comparisons

Table 10: **Endpoint and behavioral release requirements select different updates.** All counts are computed on our 17 frozen release populations, not taken from the cited papers. Agreement is with the complete-census endpoint-plus-handoff contract. The last two columns count contract-incompatible releases accepted and contract-compatible releases withheld under that reference contract.

Acceptor	Contract agreement	Incompatible accepted	Compatible withheld
PACE	$9/17$	6	2
SGM product rule	$12/17$	0	5
Hoeffding endpoint gate	$8/17$	3	6
Empirical-Bernstein gate	$9/17$	2	6

E.1 Exact finite-population computation

For a fixed label population of size M, enumeration covers all nonnegative triples summing to M, a total of $(M+1)(M+2)/2$ possibilities. The compatible set is determined by the predeclared contract. Likelihoods are calculated in log space without truncation or an asymptotic approximation. An ordered prefix and its count-vector multivariate hypergeometric likelihood differ by $t!/\prod_c s_{t,c}!$, common to every population total. This factor cancels in the mixture-to-maximum ratio. Completion checks precede statistical evaluation and handle empty feasible classes. Censuses performed after a decision validate it without contributing evidence to the original stopping time.

Table 11: Matched acquisition comparisons for repaired artifacts. Counts exclude mandatory endpoints, development, and post-decision verification.

Repair	Risk level	Census	Completion only	Sequential
F3	0.025	62	49	24
F2 (original panel)	$1/120$	61	54	46
F4	$1/240$	65	57	50

These comparisons hold the record table and acquisition order fixed. In a fixed SciFact population with 26 labels, 200 predeclared orders yield mean sequential acquisition 16.44, versus 20 for com-

pletion checks. The statistical boundary certifies in 179 orders at $\alpha = 0.025$. The contract requires at least 20 non-tied comparisons in the full population and $P > H$; it does not require observing 20 labels before certification.

In the 13 original prospective audits, completion checks alone determine the decisions. Acquisition totals are 103/131 in the fixed-executor cohort and 66/101 in the same-backbone cohort. For the latter, acquired counterfactual executions consume 204,533 tokens versus 347,683 for a census, a 41.2% reduction. Across nine endpoint-compatible cross-system cases, the sum of per-case median acquisition counts is 100.5 against a total census of 505. This is an aggregate of medians, not a fractional count in one execution. These figures describe audit acquisition rather than end-to-end repair cost.

E.2 Different acceptance targets

The 16-scenario release-rule comparison includes five complete-contract-compatible and eleven incompatible populations. Mean-only and endpoint negative-flip rules use endpoint performance; strict action agreement requires no departure; charging every migration assigns a violation to every departure. The preservation-only rule uses middle outcomes but omits the benefit clause. The full contract supplies the reference verdict. The negative-flip gate accepts four incompatible populations;

Table 12: Agreement with the full behavioral contract on 16 frozen scenarios.

Rule	Agreement
Mean-only gate	$8/16$
Endpoint negative-flip gate	$12/16$
Strict action agreement	$11/16$
Charge every migration	$13/16$
Preservation-only with middle outcomes	$14/16$
Complete contract (reference)	$16/16$

strict agreement rejects all five compatible ones. Charging every migration rejects one compatible and accepts two incompatible populations. Preservation-only accepts two incompatible populations. The separate 17-population named-rule comparison is reported in Table 10. These results quantify acceptance-target differences. They do not claim that an endpoint-oriented method violates its native guarantee or that its complete optimizer was rerun.

F Cross-System Evidence and Robustness

F.1 Update coverage and representative complete-record results

The 17 releases in Table 13 comprise six evidence-QA replacements, three code-generation updates, three AppWorld upgrades, two retrieval routers, and three agent or policy updates. They total 9,291 evaluation units. Component replacement and model-upgrade rows have no independent repair proposer. The AgentDojo component update replaces GPT-4.1 with GPT-5.2. ToolSandbox replaces Qwen3-4B with Qwen3-8B.

Representative outcomes explain the complementary clauses. Each NQ router update has one beneficial and 13 harmful departures while passing the endpoint clause. The FEVER producer has 80 beneficial departures out of 99 but 124 endpoint regressions in 2,000 tasks, exceeding its 5% budget. The HotpotQA producer has 20 beneficial departures out of 21 and satisfies the complete contract. SciFact producer and verifier updates have 20/26 and 22/31 beneficial departures and both pass. The fixed-backbone ALFWorld component study solves 263/354 tasks versus 253 for the incumbent, with ten beneficial departures and no endpoint regressions. It is separate from the 200- and 134-task feedback experiments.

F.2 Replay and implementation checks

Fresh replay reproduces 40 of 43 original labels. Applying the predeclared three-draw majority rule to disputed cases preserves the two associated release decisions. A separate outcome-blind

Table 13: **Model and update coverage of the complete-record panel.** The six backbones are Qwen3-4B, Qwen3-8B, GPT-4.1, GPT-5.1, GPT-5.2, and GPT-5.4. Update sources are specified explicitly; frozen component replacements and model upgrades do not require a separate proposer.

Systems	Releases	Units	Update source and execution configuration
Evidence QA	6	6,818	HotpotQA, FEVER, and SciFact producer/verifier replacements: Qwen3-8B → GPT-4.1, with a fixed GPT-5.4 downstream controller.
Code generation	3	920	MBPP+: Qwen3-8B → GPT-4.1. HumanEval+ and MBPP+: GPT-5.1 → GPT-5.4. LLMs generate and select code; EvalPlus executes tests.
AppWorld	3	180	Agent-model upgrades: GPT-5.1 → GPT-5.2, GPT-5.1 → GPT-5.4, and GPT-5.2 → GPT-5.4.
Retrieval routing	2	480	NQ: inductive MF → best-single or embedding-logistic routing, using the same public bank of predictions from 15 candidate RAG models.
AgentDojo, ToolSandbox, ALFWorld	3	893	AgentDojo: GPT-4.1 → GPT-5.2. ToolSandbox: Qwen3-4B → Qwen3-8B. ALFWorld: researcher-specified typed policy, with Qwen3-8B proposal/serialization and a fixed Qwen3-8B backbone.
Total	17	9,291	

repeat-control panel with 54 units preserves all nine panel decisions under majority labels. These checks examine stability of the execution evidence; they are distinct from the conditional sampling guarantee.

Implementation validation includes 2,600 acquisition orders on 13 empirical tables and 3,000 orders on six synthetic finite populations. All 5,600 agree with the complete-census verdict. This is a finite implementation check, not an empirical proof of zero error under every order. The risk bound follows from Appendix B.3 under its stated conditions.

F.3 Tie treatment and contract sensitivity

The recorded $P/H/T$ counts permit exact re-evaluation of the basic contract without rerunning the models. Holding the 159-task workloads and strict 10% endpoint budget fixed, replacing the $H+T$ clause by an H-only clause with the same tolerance produces the following decisions: All these

Table 14: Contract sensitivity derived from complete-record counts. C: compatible; W: incompatible. These are census verdicts under the specified alternative contract, not rerun sequential certificates.

Experiment	Artifact	H	$H+T$	$H+T \le 15$	$H \le 15$
Original panel F1	Original	7	20	W	C
	Repair	11	18	W	C
Original panel F2	Original	9	21	W	C
	Repair	2	8	C	C
Original panel F4	Original	14	22	W	C
	Repair	5	7	C	C
Matched F2	Endpoint + trace	10	21	W	C
	CounterGraft	3	9	C	C

artifacts satisfy the endpoint clause. Their differing release decisions under the primary contract therefore depend on its treatment of ties. The reduction in harmful replacements remains visible independently: 23 to seven across original-panel F2/F4 and ten to three in matched F2. A harmful-only budget and a strict-improvement budget encode different requirements, and certification is relative to the chosen requirement.

For an integer violation count K, a strict rate clause passes exactly when $\epsilon > K/N$. In matched F2, the unearned-change frontiers are $21/159$ and $9/159$; thus any common tolerance in

$(9/159, 21/159]$ separates these artifacts on that clause. The stated 10% tolerance lies within this interval. Projection changes, by contrast, can change which records enter the departure population and require the corresponding behavior signatures. They cannot be reconstructed from aggregate counts alone.

Benchmark sources. Evidence-QA components use HotpotQA, FEVER, and SciFact (Yang et al., 2018; Thorne et al., 2018; Wadden et al., 2020). Code generation uses EvalPlus evaluation (Liu et al., 2023); interactive-agent updates include AppWorld and ToolSandbox (Trivedi et al., 2024; Lu et al., 2025). The retrieval panel uses RAGRouter's public prediction bank (Zhang et al., 2026).

G Matched AgentDojo Repair Protocol

Experimental unit and controlled configuration. The comparison uses one initial artifact, the original F2 successor, in both feedback conditions. The proposer, executor, editable policy and prompt code, development workload, and proposal budget are identical. The proposer is `gpt-5.4-2026-03-05` at temperature zero, with at most eight editing steps per opportunity. GPT-4.1 executes candidates at temperature zero; thinking is disabled and the maximum output is 4,096 tokens. The editable files are `policy.py` and `prompts.py`. The experiment compares feedback for free-form code revision. It is a separate experiment from the earlier three-target repair panel; its final artifacts and confirmation results are not replacements for that panel's F2 records.

Feedback and opportunity budget. The baseline proposer receives endpoints, selected actions, tool calls and returns, complete execution and failure traces, and costs. CounterGraft adds child-on-parent outcomes, traces, and replacement labels. Each arm runs all 16 proposal opportunities in two batches of eight, regardless of early qualification. Invalid, duplicate, or unloadable proposals and runner failures consume opportunities and remain in the ledger.

Common selection rule. Both arms use the same predeclared lexicographic selector. It first prioritizes candidates satisfying the full development contract on the fixed 64-task workload: positive endpoint gain, at most six endpoint regressions, and at most six unearned replacements. It then maximizes task success, minimizes unearned replacements, minimizes endpoint regressions, and breaks remaining ties by earlier opportunity index. Consequently, behavioral evaluation is available to the selector in both conditions. The intervention is whether this counterfactual information is also supplied to the proposer for revision; it does not compare different release targets or selection objectives.

Freezing, confirmation, and verification. One artifact per arm is frozen before opening the same previously unseen, prelocked 159-task confirmation workload; neither runner-up substitution nor post-confirmation revision is allowed. Both pass the endpoint regression clause: endpoint-and-trace repair incurs two regressions and CounterGraft incurs one. The audit statistically certifies CounterGraft and deterministically withholds the baseline. Full censuses are performed only after these decisions to verify the outcomes; they do not enter development, candidate selection, or certification.

Complete outcomes. The baseline solves 156 tasks and has 37 beneficial, ten harmful, and eleven tied departures, giving 58 departures and 21 unearned changes. CounterGraft solves 157 and has 48 beneficial, three harmful, and six tied departures, giving 57 departures and nine unearned changes. Unearned changes decrease by 12 (57.1%), and harmful changes decrease by seven (70%), while the departure count differs by only one. These are counts for the selected artifacts from this single matched target, not estimates averaged across independent repair runs.

Confirmation contract and stopping receipt. The contract requires $L/159 < 0.10$ and $(H + T)/159 < 0.10$, allowing at most 15 violations on each clause. The acquisition permutation is bound before labels are opened. For CounterGraft, $M = 57$ and $\alpha = 0.025$, giving a certification threshold of 40. The fixed mixture is uniform over the 136 compatible population triples with $H + T \leq 15$. At draw 34, the observed counts are $(P_{34}, H_{34}, T_{34}) = (30, 2, 2)$, yielding

$$E_{34} = 42.7181229 > 40, \qquad \log E_{34} = 3.7546233.$$

The preceding prefix $(29, 2, 2)$ gives $E_{33} = 25.5902487 < 40$; the next beneficial record therefore crosses the statistical boundary. The audit stops with 23 continuations unqueried, a 40.4% reduction relative to a complete census. Deterministic completion cannot yet certify: four observed unearned labels plus 23 unresolved records can exceed the budget of 15. The post-decision census is $(48, 3, 6)$, with remaining counts $(18, 1, 4)$, and confirms the contract verdict. For the baseline, acquisition 44 of 58 reveals its sixteenth unearned label, making the contract impossible under every remaining completion and triggering deterministic withholding. Its post-decision census is $(37, 10, 11)$. Neither census contributes to the stopping decision. Supplementary reference code reproduces the two CounterGraft evidence values by exact enumeration; these receipts belong to the matched experiment, separately from the original-panel acquisition results in Section 4.4.

Proposer constraints and feedback instruction. Both arms must return a complete, loadable policy/prompt artifact, edit only the two declared files, and infer general control rules rather than hard-code task identifiers. They cannot access confirmation identities or outcomes, or modify the evaluator, utility, projection, or contract. The CounterGraft-specific instruction is:

> Beneficial departures provide evidence for retaining a behavioral change. Harmful and tied departures identify changes that may need revision, reversion to the incumbent behavior, or conditioning on task state. Use the execution traces to infer a general control rule; do not hard-code evaluation identities.

G.1 Resource cost of matched repair

Both arms receive 16 proposal opportunities and compute development counterfactual labels for the common selector. Table 15 summarizes measured input-plus-output tokens. Through the release decision, CounterGraft uses 3.1% more tokens while reducing harmful replacements from ten to three. Including verification census, the difference is 2.7%. These are token counts, without model-specific price or caching adjustments.

Table 15: Matched F2 resource use in millions of input-plus-output tokens. Development includes endpoints and child-on-parent execution; confirmation includes endpoints and replay up to the decision. Census is additional post-decision verification. Totals are computed before rounding.

Stage	Endpoint + trace	CounterGraft
Proposal generation	0.615	0.341
Development execution	23.835	23.982
Confirmation to decision	3.361	4.349
Total through decision	27.811	28.673
Additional verification census	0.131	0.034
Total including census	27.941	28.707

Reported development elapsed time is 51m29s versus 84m31s; confirmation through the decision takes 40m15s versus 44m29s. Thus near-matched token usage does not imply equal latency. Acquisition stops after 44/58 versus 34/57 continuations, leaving 14 versus 23 for verification. These different artifacts have different execution lengths; replay counts alone do not measure their token costs. Figure 4 separately compares sequential and deterministic checks on the same artifacts and orders.

H Cross-Model Repair Method Comparisons

H.1 Adaptations and common selection

Parent is the incumbent reference; repair methods start from the same initial successor for each target and executor. GPT-5.4 proposes for all three AgentDojo executor settings. Each method–target pair receives one search, with three targets per executor. Methods edit only `policy.py` and `prompts.py`, use the same 64-task development workload, and receive at most five batches of eight opportunities. Invalid proposals consume opportunities. These are common caps, not matched realized token budgets.

Adapted Self-Refine (Madaan et al., 2023) retains iterative self-feedback and revision. Adapted GEPA (Agrawal et al., 2026) retains trajectory reflection, a candidate pool, and mutation/merge. Adapted HSI (Zhou, 2026) retains hierarchical harness revision and archive maintenance. Original HSI uses one frozen model across execution and self-modification; our AgentDojo adaptation uses the common GPT-5.4 proposer and declared executor, and standardizes the editable surface, workload, budget, and confirmation. Internal proposal mechanisms differ; final export uses the common selector below. Baseline proposers receive ordinary endpoint outcomes and execution context, including traces, tool returns, and failures. CounterGraft additionally receives child-on-parent outcomes and replacement labels.

The independent evaluator computes complete development comparisons for every executable candidate. Qualification requires greater success than the incumbent, $L/64 < 0.10$, $J/64 < 0.10$, and any pre-enabled directional clauses. After completing a batch, search stops if at least one candidate qualifies. Qualified candidates in that batch are ranked by success gain, then smaller J, smaller L, and earlier opportunity index. If none qualifies after five batches, the same ranking selects the best failed candidate. Counterfactual evidence is available to the common evaluator for eligibility, stopping, and final selection in every arm; its outcomes and labels are withheld from baseline proposers. The comparison measures proposal methods under shared selection infrastructure.

Exactly one artifact per method–target pair is selected and cryptographically frozen before confirmation. Neither proposer nor development selector accesses confirmation outcomes. Failure permits no runner-up substitution, further revision, or reselection in that release opportunity. AgentDojo confirmation uses 159 tasks per target and allows at most 15 regressions and 15 unearned changes. Post-decision censuses support reporting and verification, not selection or original certification.

H.2 Raw counts and behavior

Each AgentDojo column in Table 16 contains 477 target–task evaluations, not 477 distinct task identities or independent searches. Percentages and gains are computed from raw integers before rounding. Certified counts are out of three artifacts. The original GPT-4.1 F1/F2/F4 repairs are the CounterGraft entries here, not an additional replication.

Table 16: Raw confirmation successes; AgentDojo certification counts in parentheses.

Method	GPT-4.1	GPT-5.2	GPT-5.4	Qwen3-8B
Parent	300/477	343/477	382/477	95/134
Adapted Self-Refine	438/477 (0/3)	423/477 (1/3)	435/477 (1/3)	97/134
Adapted GEPA	447/477 (1/3)	440/477 (1/3)	447/477 (2/3)	99/134
Adapted HSI	451/477 (1/3)	446/477 (2/3)	452/477 (2/3)	100/134
CounterGraft	457/477 (2/3)	455/477 (3/3)	459/477 (3/3)	101/134

H.3 ALFWorld development yield and confirmation

Qwen3-8B proposes and executes over the common module pool and fixed development split. Each method runs all 12 opportunities, applies the frozen selector, and freezes one artifact before opening 134-task confirmation. All methods produce 12 executable candidates. Development-qualified counts are 1, 3, 5, and 8 for adapted Self-Refine, GEPA, HSI, and CounterGraft, respectively. These are development opportunity yields, not confirmation certification rates. The contract requires $L \leq 6$, $H + T \leq 6$, $P > H$, $P + H > 0$, and strictly greater success than the incumbent.

H.4 Relationship to controlled feedback experiments

Complete-method comparisons differ from the fixed 16-opportunity matched F2 experiment in Appendix G, which holds the search setup and selector fixed and changes only proposer-visible feedback. The earlier three-arm ALFWorld feedback study in Appendix D is also separate; its endpoint and regression-augmented arms are not renamed as Self-Refine, GEPA, or HSI. Table 17 collects original-to-repair and controlled-feedback outcomes.

Table 17: **Executable improvement and controlled feedback across two model configurations.** A/B use a GPT-5.4 proposer and GPT-4.1 executor on 159 tasks; C uses Qwen3-8B for both roles on 200 tasks. A includes all three held-out targets; B is a separate 16-opportunity-per-arm comparison. L counts regressions and $J = H + T$. Budgets are 15 for A/B and nine for C, which also requires $P > H$ and $P + H > 0$. Shading identifies CounterGraft; bold success counts mark pairwise improvements.

Target	Program / feedback	Success ↑	L ↓	$P/H/T$	J ↓	Contract
A. AgentDojo: repairing high-scoring successors						
F1	Original	150	4	46/7/13	20	Fail
	CounterGraft repair	149	7	52/11/7	18	Fail
F2	Original	148	8	46/9/12	21	Fail
	CounterGraft repair	**154**	3	53/2/6	8	**Pass**
F4	Original	149	7	43/14/8	22	Fail
	CounterGraft repair	**154**	2	58/5/2	7	**Pass**
B. AgentDojo: same initial F2, code space, and proposal budget						
F2	Endpoints + traces	156	2	37/10/11	21	Fail
	CounterGraft feedback	**157**	1	48/3/6	9	**Pass**
C. ALFWorld: matched module-level feedback under Qwen3-8B						
	Module-level endpoints	152	2	2/3/20	23	Fail
	CounterGraft feedback	**153**	0	2/0/0	0	**Pass**

H.5 Repair trajectories and prospective installation

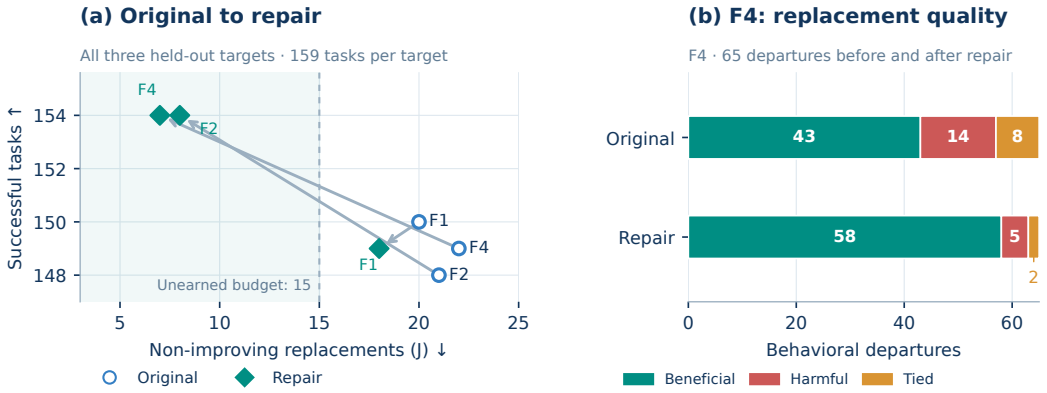


Figure 3: **Repair improves task success and replacement quality.** (a) Original-to-repair transitions for all three held-out targets on the same 159-task workload. Shading marks the non-improving-replacement budget; all six artifacts pass the endpoint clause. (b) F4 retains 65 departures while their composition improves. Non-improving replacements comprise harmful and tied departures. Both versions use the fixed GPT-4.1 executor.

Table 18: **Certified successors actually inherit in both prospective configurations.** GPT-5.4 proposes in both cohorts. All audited candidates pass endpoint regression; only contract-compatible programs are installed. Entries report candidate counts.

Executor	Audited	Higher score	Endpoint pass	Contract pass	Installed
Fixed GPT-4.1	5	5	5	1	**1**
Same-backbone GPT-5.4	8	6	8	1	**1**

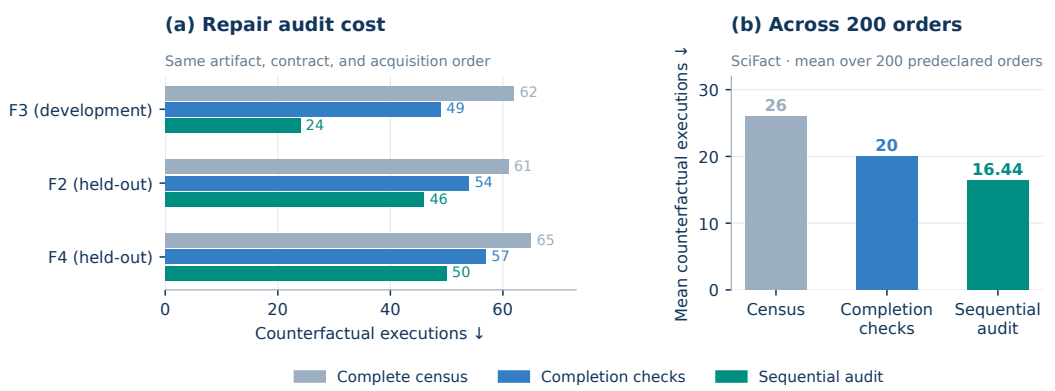


Figure 4: **Sequential evidence reduces acquisition beyond completion checks.** (a) Matched comparisons for development repair F3 and held-out repairs F2/F4. (b) Means over 200 predeclared orders of the fixed SciFact table. Counts include counterfactual executions up to the decision, excluding mandatory endpoints, development, and post-decision verification. Risk levels and contract conditions are given in the text.