WHAT NET ACCURACY HIDES: AUDITING REPAIR AND BREAKAGE IN VERIFY–REFINE WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

A small accuracy change can hide substantial changes in which problems an LLM workflow solves—and regressions may be attributed to refinement that never executed. We present an execution-path audit for verify–refine workflows that pairs baseline and workflow outcomes with the initial answer's source, verifier decisions, and calls admitted or refused by the budget. Across program synthesis and mathematics, these records distinguish answer regressions from the execution paths on which they occur. In a GPT-4o study, a code workflow changes accuracy by -3.3 percentage points (95% CI: $[-8.7, +2.0]$), while repairing 24.4% of baseline failures and breaking 13.8% of baseline successes. Under tighter budgets, 96 of 450 executions stop before any refinement runs: a wrong final answer in these runs cannot be attributed to an executed refinement. With equal refinement allowances and no blocked suffix calls, reusing a stored baseline output yields $5/347$ breakages versus $27/347$ for same-policy regeneration on one mathematics reference set. A second frozen reference set on the same tasks repeats the breakage ordering, but not the net-accuracy advantage. A serving control changes 8.2% of paired workflow outcomes under batching while net accuracy moves by only -0.22 percentage points. These findings support reporting repairs, breakages, and actual execution paths together, and evaluating preservation against the repairs it forgoes.

1 INTRODUCTION

A verify–refine workflow can gain little in net accuracy while changing many answers. In our GPT-4o code study, the workflow repairs 24.4% of a baseline's failures but breaks 13.8% of its successes; the resulting accuracy difference is -3.3 percentage points, with a 95% interval spanning zero. These opposing transitions matter when previously solved tasks must remain solved. Their counts alone, however, do not explain the failure: a wrong final answer may follow an accepted wrong draft, an unsuccessful revision, or a budget refusal before revision even starts.

Correct-to-incorrect transitions are established measures in model-update regression studies (Yan et al., 2021; Xie et al., 2021), and self-correction work already separates them from successful corrections. Our question concerns the executed workflow: *which calls and incumbent choices produced these transitions, and which planned calls never ran?* This requires pairing outcome labels with execution records. In particular, a correct baseline answer does not imply that an independently generated workflow draft was correct, and the same model, prompt and decoding policy do not guarantee the same answer.

We present an execution-path audit that joins four records for each task and execution seed: the baseline and workflow outcomes, the source of the initial answer, the verifier's decisions, and the budget ledger for attempted calls. The audit distinguishes accepted wrong incumbents, failed rejection paths, and failures before verification, then records whether refinement actually executed. It also makes the starting artifact an explicit intervention: assign the stored baseline output, regenerate it under the same declared policy, or change the drafting policy before a shared verify–refine suffix (Figure 1). Assignment uses an available baseline artifact, not a known-correct answer; correctness is used only for external evaluation.

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.

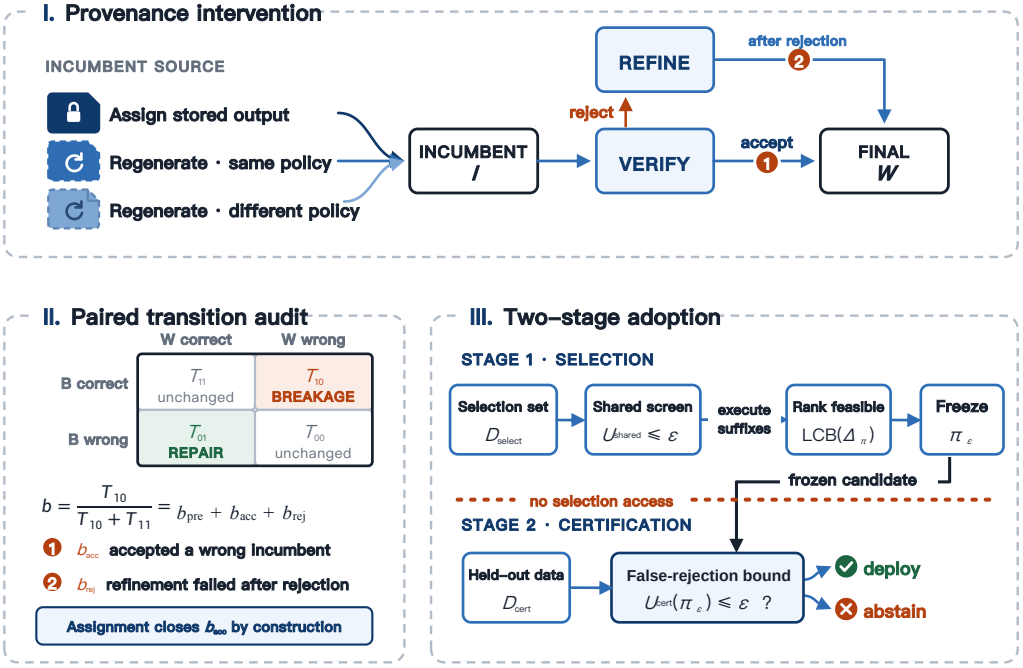

Figure 1: The executed design. The incumbent is assigned from the stored reference, regenerated under the same declared policy, or regenerated under a deliberately different policy, before a common verify–refine suffix. Assignment against same-policy regeneration is the provenance comparison; different-policy regeneration substitutes a different drafting node of the same model. Panel II resolves breakage into the three terminal paths the audit reports: b_{pre}, when the budget refuses the call before the verifier ever runs and the workflow returns an incorrect fallback; b_{acc}, an incumbent the first verifier accepts outright; and b_{rej}, the trajectory after a first rejection, including refinement the budget blocks. Only b_{acc} is closed by construction, and only under assignment. Panel III is a secondary adoption case study, not a third audit mechanism: a frozen two-stage selection-and-certification protocol; Appendix C defines its risk bounds, and Appendix F reports the held-out decision.

The audit changes how the results should be read. In the GPT-4o code study, tighter budgets lower repair and raise breakage, but 96 of 450 executions stop before any refinement runs. Those failures cannot be attributed to a refiner damaging its input. Masking visible tests changes both verifier decisions and exposure to refinement, raising breakage from .138 to .235. On a local Qwen2.5-Coder-7B-Instruct deployment, batching flips 37 of 450 final workflow outcomes—19 losses and 18 gains—while net accuracy moves by only -0.22 percentage points. The paired records reveal the transitions that cancel in the aggregate.

The provenance intervention asks whether regenerating a baseline answer is interchangeable with retaining it. Equal total budgets confound this question because regeneration spends an extra call before refinement. We therefore fund drafting separately in a GLM-4-Flash follow-up and give both arms equal refinement allowances, with no suffix calls blocked. On two frozen mathematics reference cohorts, assignment yields $5/347$ versus $27/347$ breakages and $7/343$ versus $15/343$. The first paired interval excludes zero; the second reaches zero. The accuracy advantage does not persist across cohorts because preservation can forgo repairs. Under a byte-stable serial serving condition, the two arms instead coincide at all 450 paired positions. The practical benefit of retaining an answer therefore depends on the executed regime.

Our contribution is an evaluation protocol and its empirical findings: paired transition counts linked to execution paths; budget-aligned tests of stored versus regenerated incumbents; and a serving intervention showing substantial outcome changes behind a small net effect. We provide derived ex-

ecution records and offline analyses to make these distinctions inspectable. Task-clustered intervals and design-sensitivity analyses delimit what the comparisons resolve, particularly when breakages are rare. The evidence supports auditing preservation alongside improvement in short-form code and mathematics workflows; it does not establish a uniformly better refinement policy or performance on repository-level agents.

2 Background and related work

Iterative refinement and its discontents. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) made the generate–critique–revise loop a default building block, and multi-agent debate (Du et al., 2023) and self-consistency (Wang et al., 2022) extended it to populations of samples; QualityFlow gates acceptance on an LLM quality check (Hu et al., 2025b), closest to our *protected* arm (Appendix V) and scored by the pass rate our audit decomposes. Huang et al. (2024) then showed that models cannot reliably self-correct reasoning without external feedback, attributing apparent gains to oracle labels leaking into the correction signal. SCoRe (Kumar et al., 2025) measures incorrect-to-correct and correct-to-incorrect transitions and trains self-correction policies to improve their balance. We use these established outcome distinctions and add the execution record: whether a verifier accepted a wrong incumbent, refinement ran, or the budget refused the planned call.

Regression under updates, and negative flips. Measuring *which* predictions a new system spoils is established for model updates: negative-flip rate is the standard measure (Yan et al., 2021), regressions are pervasive across NLP update settings (Xie et al., 2021), and structured prediction needs its own measures (Cai et al., 2022). That literature replaces the model and counts flips; classical regression-test selection (Leung & White, 1989) changes the program and selects tests. Closest to us, QiMeng-PRepair reduces over-editing by training repair models to retain correct program context (Ke et al., 2026); we instead audit inference-time correctness flips and their control-flow paths without training. We keep the model weights, the task set and the downstream suffix fixed while intervening on how the incumbent artifact is produced, and extend the comparison to which execution path produced each flip and what the budget allowed to run. Our two regeneration arms are weaker perturbations than a model update — a different drafting node in the same model, and the same drafting policy re-invoked — so they measure regressions induced by changing incumbent production before a fixed downstream suffix.

Automated workflow design and test-time compute. Search systems vary prompts, topology, modules, and model/cost configurations (Zhuge et al., 2024; Hu et al., 2025a; Zhang et al., 2025b; Shang et al., 2025; Zhang et al., 2025a), and allocation methods trade compute against verification granularity or reuse prefixes (Snell et al., 2024; Chen et al., 2026; Li et al., 2026b;a). Our audit is orthogonal to the objective they optimize: it records damage to correct references and the path causing it, which a net compute–accuracy frontier hides. Our primary-paper and official-repository audit (Appendix G) finds no named first-class operator that assigns the reference system's stored output; ADAS can express the behavior through arbitrary generated code, so the claim is about the documented search vocabulary, not inexpressibility. Repository-level benchmarks such as SWE-bench (Jimenez et al., 2024) make the execution environment part of the evaluation object; our program-synthesis tasks isolate test-gated edits and their paired outcomes, and establish nothing about repository or tool state in a long-running agent.

Abstention and conservative updates. Declining to act on a low-confidence prediction is old: the reject option (Chow, 1970) reappears as selective classification (Geifman & El-Yaniv, 2017) and learning to defer (Madras et al., 2018; Mozannar & Sontag, 2020); cascades (Viola & Jones, 2001; Chen et al., 2023) gate expensive stages on a cheap one; conservative updates bound a candidate against a baseline (Thomas et al., 2015; Garcelon et al., 2020). We present neither guarded fallback nor the concentration inequality as new; the new empirical question is whether a generated workflow actually preserves the baseline artifact it claims to protect.

Evaluation. We use MBPP+ and HumanEval+ (Austin et al., 2021; Chen et al., 2021) in the rigorous-test formulation of EvalPlus (Liu et al., 2023), MATH (Hendrycks et al., 2021) and BIG-Bench Hard (Suzgun et al., 2023); multi-fidelity screening follows successive halving (Jamieson &

Talwalkar, 2016; Li et al., 2018), and intervals are stratified paired bootstrap (Efron et al., 2000). All matrices report task-clustered descriptive 95% intervals without a familywise significance claim.

3 THE AUDIT

Let B be a baseline and W a workflow on the same tasks. Write $p = \Pr[B \text{ correct}]$ and

$$r = \Pr[W \text{ correct} \mid B \text{ incorrect}], \qquad b = \Pr[W \text{ incorrect} \mid B \text{ correct}], \tag{1}$$

so that $\Delta = \Pr[W \text{ correct}] - \Pr[B \text{ correct}] = (1-p)r - p\,b$, an exhaustive algebraic identity for binary outcomes. Its role is to fix what the audit must measure, not to be a result. For $r > 0$ a strict gain requires $b/r < (1-p)/p$, which tightens sharply as the baseline improves (thresholds $27/73$ at $p = .73$ and $2/23$ at $p = .92$). We average joint event indicators within task, then across tasks; conditional rates divide these averaged numerators by their averaged eligibility indicators, rather than averaging task-specific conditional rates. In complete three-seed matrices this equals pooling eligible pairs. Intervals resample tasks with their seeds, recomputing the ratios (Appendix A); 450 executions are at most 150 independent units.

Every workflow here runs under a reservation ledger; before each model call the executor reserves that call's worst-case cost against a per-task cap and refuses it if it does not fit. A refused call is neither a wrong answer nor a slow one; a planned step *did not execute*, and the task ends on the fallback. A wrong fallback on a reference-correct task is a breakage, even if no refiner ran. Reference correctness does not establish that the workflow's independently generated incumbent was correct. We therefore report first-verifier rejections, how many had a correct stored reference, and how many reached refinement versus were blocked. A comparison whose arms differ in blocking is an *operational* comparison of whole workflows, and we label it as such.

Consider a workflow that emits an incumbent I, returns it when the verifier accepts, and otherwise produces a rejection-path output J. Two events must be kept apart: whether the verifier executed at all, and, if it did, whether it rejected. Writing V for "the verifier executed", R for rejection, and J_{pre} for whatever the fallback returns when the budget refuses the call before verification,

$$\begin{aligned} b = {} & \underbrace{\Pr(\neg V,\ J_{\text{pre}} \text{ wrong} \mid B \text{ correct})}_{\text{never verified}} + \underbrace{\Pr(V, \neg R,\ I \text{ wrong} \mid B \text{ correct})}_{\text{accepted wrong incumbent}} \\ & + \underbrace{\Pr(V, R,\ J \text{ wrong} \mid B \text{ correct})}_{\text{refinement failed or never ran}}, \end{aligned} \tag{2}$$

The corresponding integer counts are K_{pre}, K_{acc} and K_{rej}. We further partition the last probability as $b_{\text{rej}} = \Pr(V, R, \neg E, J \text{ wrong} \mid B \text{ correct}) + \Pr(V, R, E, J \text{ wrong} \mid B \text{ correct})$, where E means at least one refinement completed execution. Each stratum records terminal refusal, execution failure or completion: E alone does not imply refinement caused the error (Appendix S). Gating refinement cannot remove the acceptance term. Setting $I = B$ is sufficient to make it identically zero: if the incumbent is the reference system's actual stored output, and not merely an answer produced under the same configuration, then B correct implies I correct. The first term vanishes too whenever the pre-verification fallback returns the incumbent. In our executions assignment never reaches this path because the initial cap admits verification; making no drafting call preserves that headroom. Under both conditions

$$b = \Pr(V, R,\ J \text{ wrong} \mid B \text{ correct}) \ \le\ \Pr(R \mid B \text{ correct}). \tag{3}$$

This is event inclusion, so the data test implementation compliance and residual rejection-path damage rather than validate an inequality. Its auditable premises: the incumbent is the stored output, acceptance returns it unchanged, every path to a different answer follows rejection or a refused call, and risk is measured on reference-correct instances. Our intervention injects the stored string with an explicit `assign` node and disables caching; same-policy regeneration is a separate arm because the same declared policy does not establish $I = B$.

4 EXPERIMENTAL SETTING

Three incumbent sources. *Assignment* copies the stored reference string without a model call. *Same-policy regeneration* makes a fresh call with the reference model, prompt and decoding policy.

Different-policy regeneration substitutes a different drafting node of the same model, here chain-of-thought solving at temperature 0.7, leaving model, tasks, seeds and the downstream workflow fixed: a within-model perturbation, not a model upgrade. These labels describe artifact provenance, not an assumed ordering of model behavior.

GPT-4o workflows. Vanilla drafts with chain-of-thought at $T = .7$ (1,536 output tokens); protected drafts the single-call baseline at $T = 0$ (code: 1,024; math: 1,536). Both stop on verifier acceptance; rejection triggers feedback-conditioned refinement at $T = .7$ (1,536 tokens). Three refine–verify back edges permit at most four refinements. Loop exhaustion or reservation refusal returns the current answer; ledger caps can stop earlier (Appendix V).

Studies and models. Each study answers a different question. (i) *What does net accuracy hide?* A GPT-4o study with a preregistered design: five claims committed and tagged (`prereg-freeze-partI`) before any frozen test split ran; 150 tasks per family, three seeds, composite `tight`/`loose` profiles, and a verifier-signal ablation. The paired transition estimand was corrected after the freeze; reported intervals are descriptive and unadjusted, not the preregistered Holm family of tests (deviation log in the supplement). (ii) *Does the verifier-sensitivity pattern hold in a second family?* A replication of all ten verifier and domain conditions through a configured Kimi K3 route (returned model identity was not retained) (Appendix O). (iii) *Does the incumbent's provenance change the transitions?* A post-hoc GLM-4-Flash matrix on the official `glm-4-flash-250414` API: two domains, two profiles, a stored reference and three incumbent interventions, 150 tasks and three paired seeds per cell, caching disabled, temperature 0.7 unless a node specifies otherwise. (iv) *Is the same declared policy the same artifact?* A serving-regime intervention on a local Qwen2.5-Coder-7B-Instruct deployment, with a serial control in which the two provenance arms coincide (Appendix Q, H). (v) *Is the same-policy null an effect or a reachability artifact?* A budget-parity follow-up that funds drafting outside the shared suffix allowance (§7). No model is fine-tuned, and profile contrasts are composite rather than the effect of one resource.

Verifiers. Two regimes, never conflated. In the code family the verifier runs the task's own visible tests, environment feedback that is part of the specification, with hidden extended tests reserved for the external grader. In math and logic no oracle is available and the verifier is a gold-free independent re-derivation agreement check. The standard verifier makes one call at $T = .3$ (1,536 tokens), without the incumbent in its prompt, and compares extracted answers. Parse failure or symbolic-check timeout counts as disagreement; budget and execution failures remain separate statuses. Gold answers never enter workflow state. A "fixed" verifier means its program, prompt, parameters and decision rule are shared across arms; its rejection rate is an outcome that can change with the incumbent source or upstream budget consumption. The two regimes differ in fidelity, scored against known incumbent labels in Appendix D.2.

Tasks. Code: MBPP+, split 120/40/150, HumanEval+ held out. Math: MATH levels 4–5 over four subjects, split 120/40/150, two further subjects held out; the GLM certificate uses 1,194 disjoint MATH tasks. Logic: BIG-Bench Hard, 120 tasks, reserved for a locked prospective prediction. Intervals use $10{,}000$ task-level paired bootstrap resamples.

Sampling temperature does not determine artifact identity. The serving configuration can also change the realized output, and we test that rather than assume it. Holding the model weights, the GPU and the decoding parameters fixed — all recorded first-draft model payloads (messages, model, temperature, output limit and seed) are identical across conditions — we vary only the vLLM serving regime of the local Qwen2.5-Coder-7B-Instruct deployment. Repeating the serial condition reproduces every first draft exactly, $450/450$, and every workflow outcome. Concurrent batching reproduces 50.0% of first drafts $[43.6, 56.4]$ and prefix caching 54.2% $[46.0, 62.2]$; excluding recovered or non-overlapping requests leaves both unchanged (Appendix Q). That greedy outputs can move under batching or reduction-order effects is established (Yuan et al., 2026; Atıl et al., 2025) and we do not claim it as a finding. What the audit adds is the paired consequence: under batching, *final workflow* correctness — not first-draft correctness — flips on 37 of 450 paired positions, 19 from correct to wrong and 18 from wrong to correct. A comparison reporting only net accuracy would report a near-zero aggregate change of -0.22 percentage points while 8.2% of paired outcomes changed. This is why the provenance of the incumbent is a separate arm rather than something a fixed policy fixes.

Table 1: Preregistered GPT-4o frozen test, 150 tasks per family, three seeds. Baselines are the strongest single-call structures; vanilla uses a capped verify–refine loop; protected replaces a temperature-zero incumbent only after rejection. Repair and breakage are conditional rates on eligible baseline-wrong and baseline-correct pairs. Paired-bootstrap 95% CIs; the intermediate `unseen` profile is in Appendix J.

Family	Budget	Workflow	Δ (95% CI)	repair	breakage	$/task
code	tight	vanilla	-0.104 $[-0.162, -0.047]$	0.203	0.220	0.0062
code	tight	protected	$+0.013$ $[+0.002, +0.031]$	0.049	**0.000**	0.0022
code	loose	vanilla	-0.033 $[-0.087, +0.020]$	0.244	0.138	0.0090
code	loose	protected	$+0.020$ $[+0.002, +0.042]$	0.073	**0.000**	0.0025
math	tight	vanilla	$+0.007$ $[-0.027, +0.040]$	0.233	0.076	0.0076
math	tight	protected	$+0.000$ $[+0.000, +0.000]$	0.000	**0.000**	0.0075
math	loose	vanilla	$+0.020$ $[-0.011, +0.051]$	0.242	0.061	0.0153
math	loose	protected	$+0.007$ $[-0.002, +0.018]$	0.042	0.006	0.0153

5 NET ACCURACY HIDES REPAIR, BREAKAGE, AND BLOCKED EXECUTION

The preregistered *protected* arm is not the later zero-call assignment intervention; it drafts its own temperature-zero incumbent and replaces it only after a rejection. Appendix V gives its initial policy, verifier, refinement schedule, iteration cap and fallback semantics.

A net effect near zero is not a workflow that changes little. At `loose`, vanilla verify–refine differs from its baseline by -0.033 $[-0.087, +0.020]$ in code and $+0.020$ $[-0.011, +0.051]$ in math, at $5.5\times$ and $2.2\times$ the cost; neither interval excludes zero and neither is an equivalence test. The decomposition carries the information: in code the same workflow repairs 24.4% of the baseline's failures and destroys 13.8% of its successes — opposing transitions that net accuracy alone does not reveal.

Budget refusal must be distinguished from executed refinement. At `tight`, code vanilla is -0.104 $[-0.162, -0.047]$. Moving `loose`→`tight`, repair falls $0.244 \rightarrow 0.203$ while breakage rises $0.138 \rightarrow 0.220$ (Figure 2, left). The ledger records 102 reservation refusals among 450 task–seed executions, covering 55 distinct tasks: 35, 34 and 33 refusals at seeds 0, 1 and 2. Of these, 96 occur before any refinement executes and six after one refinement (Appendix S). Among the cell's 72 breakages, 45 terminate at a refused first refinement and 2 at a later refusal after an executed refinement. The remaining 25 complete on initial verifier acceptance of a wrong draft, so $72 = 45+2+25$. The 45 executions blocked before their first refinement retain a wrong initial draft despite a correct paired reference; they are not damage caused by refinement. The record cannot determine what the refused call would have produced. Dollar caps do not bind; calls and output-token caps govern admission. Profiles change several caps together, so this is a comparison of composite budgets, not a single-resource effect.

Verifier signal moves breakage. Masking visible tests at $1.0 \rightarrow 0.5 \rightarrow 0.0$ moves the code/`loose` effect non-increasingly, -0.033, -0.033, -0.118 $[-0.178, -0.058]$, with breakage climbing $0.138 \rightarrow 0.235$ (Figure 2); masking changes only what the verifier sees, not the model, budget or task set. Decomposing by terminal path says which term moved (Appendix T). At full signal the verifier rejects 51/327 of reference-correct pairs and 25/276 of the accepted ones are already wrong; with no tests it rejects 327/327, so the acceptance path empties, every correct reference is exposed to refinement, and 77/327 of those refinements end wrong. The rise is therefore a change in exposure, not in silent acceptance — a composition a net-accuracy report cannot show, though it does show the 11.8-point drop itself.

Kimi K3 replicates all ten verifier and domain conditions at 100% call completion and repeats the ordering, breakage rising $0.000 \rightarrow 0.060 \rightarrow 0.216$ as visible tests are masked. On the 68 out-of-distribution tasks the signal-restored condition covers, where both use the same 189 reference-correct pairs, breakage is 0.132 without tests against 0.011 with them restored — separate runs on a matched subset, not a within-run manipulation. The pattern replicates; the magnitudes do not. The preregistered C4 outcome criterion does not isolate a mechanism: both compared workflows share

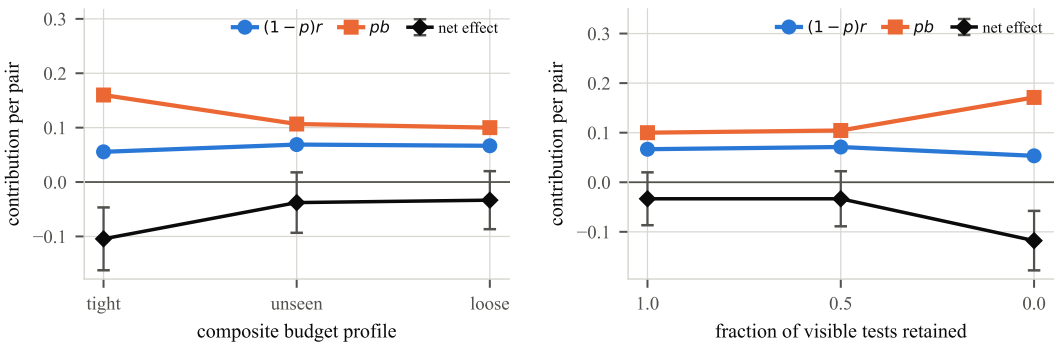

Figure 2: Repair and breakage contributions to net accuracy, both measured per paired execution: $(1-p)r$ and pb. Their difference is the net effect; unlike the conditional rates, their equality is the zero-effect threshold. **Left:** tightening the composite budget increases the excess of breakages over repairs. Dollar caps do not bind; admission failures arise from call and output-token caps (Appendix J). **Right:** withholding visible tests changes verifier decisions and exposure to refinement. With no tests the verifier always rejects, and the net effect is -0.118 $[-0.178, -0.058]$. Model, budget and task set are fixed. Both panels use frozen code test pairs; error bars are task-clustered 95% intervals on the net effect.

Table 2: Clean GLM-4-Flash provenance matrix. Repair and break are event counts over eligible reference-wrong and reference-correct pairs; accept is acceptance-path breakage; R/R_B is first rejections / rejections of correct references; E/G is initial refinements executed/blocked. The censored math/`tight` cell and costs are in Appendix D.

cell	incumbent source	Δ [95% CI]	repair	break	accept	R/R_B	E/G
code/tight	assign	$+.038$ $[+.013, +.069]$	17/138	0/312	0	97/3	97/0
	same-policy	$+.027$ $[+.004, +.053]$	13/138	1/312	1	95/3	95/0
	different-policy	$-.060$ $[-.111, -.009]$	23/138	50/312	22	137/59	82/55
code/loose	assign	$+.033$ $[+.009, +.064]$	16/136	1/314	0	95/3	95/0
	same-policy	$+.036$ $[+.011, +.064]$	17/136	1/314	0	94/6	94/0
	different-policy	$-.060$ $[-.116, -.004]$	25/136	52/314	30	136/62	136/0
math/loose	assign	$+.013$ $[-.013, +.042]$	14/107	8/343	0	125/42	125/0
	same-policy	$+.016$ $[-.009, +.042]$	21/107	14/343	0	118/47	99/19
	different-policy	$+.004$ $[-.038, +.047]$	37/107	35/343	0	143/66	126/17

the verifier gate, and their drafting policies differ. C5 is unsupported. We therefore test incumbent provenance explicitly in the intervention below (Appendices O, L, M).

6 Where breakage enters: a provenance intervention

The primary matrix contains 450 matched task–seed pairs per arm and cell on the official GLM-4-Flash API; Appendix D reports its status and control-flow audit, and Figure 3 plots it in the repair/breakage plane.

A drafting-policy shift produces operational regressions. Different-policy regeneration breaks $50/312$, $52/314$ and $35/343$ correct references against assignment's $0/312$, $1/314$ and $8/343$. These are whole-workflow contrasts: code/`tight` has 55 initial refinement gates under different-policy regeneration versus none under assignment, and math/`loose` has 17 versus none. They cannot isolate drafting provenance from reachability. The 22 and 30 acceptance-path breakages in code do localise regressions that occur without refinement. A separate three-arm study gives paired breakage differences of $+0.094$ $[0.049, 0.146]$ in code and $+0.070$ $[0.038, 0.105]$ in math (Appendix P); the model is unchanged, but only the budget-parity study below equates the suffix allowance.

Table 3: Budget-parity follow-up on the two frozen mathematics cohorts; the code replication is in Appendix U. Both arms share per-call limits and an eight-call verify–refine allowance; regeneration gets one extra drafting allowance, after which its suffix headroom matches assignment's, and no suffix call is gated. Counts are over reference-correct pairs; repair intervals are in Appendix R. Intervals are post-hoc descriptive and task-clustered.

		asg./same-policy		asg. − same-policy, 95% CI	
cohort	n	break	accept	breakage	accuracy
prior-`tight`	347	5/27	0/4	−.063 [-.099,-.031]	+.040 [+.007,+.076]
prior-`loose`	343	7/15	0/1	−.023 [-.049,+.000]	−.004 [-.031,+.022]

Stored assignment has 0 acceptance-path breakages across all four cells, as the contract requires, and in the three uncensored cells its repair/break counts against same-policy are $17/0$ versus $13/1$, $16/1$ versus $17/1$ and $14/8$ versus $21/14$, which the paired intervals do not resolve. That is a statement about this matrix, which funds regeneration's drafting call from the same envelope as its own suffix, so a null here mixes provenance with reachability; §7 removes that and concludes otherwise. The E/G column understates the mixing: replaying the original caps, code/`tight` same-policy would have been refused a planned call at 71 of 450 positions against assignment's 11 — a later-iteration asymmetry in a cell whose zero initial gates read as clean (Appendix U).

Bytes change; correctness rarely does. First-accepted byte identity ranges from 23.8% to 94.9%, yet of the 354 byte-changing pairs only one breaks a correct reference (Appendix D.1): byte instability alone does not establish a practical benefit for assignment.

One cell measures blocking, not provenance. In math/`tight`, same-policy is blocked before verification in 350 pairs and assignment executes only six of its 120 possible initial refinements: unequal path access under the ledger, and the only cell where the K_{pre} term of Eq. 2 is non-empty (Appendix D). We exclude it from every provenance and loss conclusion; math/`loose` is intermediate, blocking $19/118$ same-policy initial refinements and none of assignment's 125.

7 Budget parity: separating provenance from reachability

With drafting funded separately and no suffix call gated (Table 3), the cohorts are no longer budget conditions, only different frozen references on the same 150 tasks: prior-`tight` and prior-`loose`. This is a post-hoc controlled follow-up and every interval is descriptive.

Assignment breaks less, but does not dominate. Assignment breaks fewer reference-correct pairs in both cohorts, $5/347$ against $27/347$ and $7/343$ against $15/343$ (Table 3), but the cohorts disagree on net accuracy: $+.040$ $[+.007, +.076]$ in prior-`tight` against $-.004$ $[-.031, +.022]$ in prior-`loose`, where assignment forgoes ten repairs and the repair interval excludes zero. Regeneration also has an extra opportunity to solve a baseline-wrong task before verification; assignment can repair it only after rejection. This structural asymmetry is part of the intervention, not equal total compute or oracle selection from two candidates. Full repair/break counts appear in Appendix R. Code parity yields $0/312$ versus $3/312$ and $0/314$ versus $2/314$ breakages; its intervals do not resolve an advantage (Appendix U).

Assignment records no acceptance-path breakage in either cohort; same-policy records four and one. Stratifying by whether the regenerated draft was itself correct localises the rest (Appendices R, D.4): regeneration fails to reproduce a correct incumbent on 25 and 18 positions, of which $19/25$ and $12/18$ end wrong, whereas assignment ends correct on $24/25$ and $17/18$ at the same paired positions.

Where the serving regime is byte-stable the arms coincide entirely; on the serial Qwen deployment all 306 first-accepted drafts match their stored references byte for byte and the arms agree at all 450 paired positions (Appendix H). Batching and prefix caching break that stability on the same deployment (§4), so the condition belongs to the serving configuration.

Conditional on a correct first incumbent, post-rejection failure is unresolved and reverses across cohorts (12.8% against 25.0%; 18.4% against 9.4%, on 32–39 rejections per arm), so we claim

no superior refinement recovery. Conditioning on reference-correct positions is fixed before either intervention and defines the preservation estimand: assignment's advantage at the incumbent stage is structural, not evidence of a better refiner. Cost is not drafting alone, since suffix paths differ (Appendix R).

8 What these contrasts can and cannot resolve

The operational matrix's same-policy contrasts all include zero while §7 resolves a large one, and both follow from one design fact. Breakage is defined only where the stored reference is correct, so a cell's 450 paired executions supply 312–343 eligible positions in 106–124 clusters, 25% of screened tasks contribute none, and the breakage indicator has within-cluster correlation .00, .25 or .39: treating the design as 450 independent trials overstates its precision severalfold. Simulating the executed design for a prespecified effect size, conditional on plug-in estimates of the event rate, correlation and eligibility profile, and across cross-arm pairing structures the design does not identify, it detects a halving of code breakage with probability at most .01 and a halving at math/`loose`'s rate with probability .36–.66; reaching 80% power against the former would need 2,400–7,100 screened tasks depending on that structure. Appendix N reports the tolerable differences per cell, the full detectability surface and the simulated type-I error at the null. These are design-sensitivity calculations, not observed power; the nulls are resolution limits, not equivalence results.

9 Limitations

The certification rule could not have passed. We froze one GLM-4-Flash candidate after a 150-task selection stage and evaluated it once on 1,194 untouched MATH tasks. Breakage is $2/835 = .0024$ and accuracy changes by $+.0034$ $[-.0008, .0084]$, but the prespecified one-sided 95% false-rejection bound is $0.07699 > .05$: the rule abstains, and floors at 0.05065 even at zero observed false rejections, needing $1,226$ tasks. This floor belongs to the frozen conservative rule, not every possible certificate; alternative bounds are sensitivities, not a changed decision (Appendices F, I).

Locked transfers of p, r, b to HumanEval+ and held-out mathematics fail (MAE 0.061, sign agreement $6/8$); magnitudes do not transfer even within GPT-4o. The code benchmark is a program-synthesis proxy; a repository agent would bind its output to a snapshot, tool state and environment we do not evaluate, and public code benchmarks carry contamination risk: a memorised task is typically solved on both sides of a pair and enters neither transition, but still inflates p and shifts our denominators. The archive's 4.7% settlement overruns are examined by no-overrun subsets in Appendix K: code conclusions persist, but the mathematics vanilla/`loose` interval then excludes zero. These conditional subsets do not replace all-task results. The GLM matrices are post-hoc, the serving intervention covers one stack, the cohorts disagree on net accuracy, and template dependence may make task-cluster independence approximate.

Fallback choice also matters. Reverting to the stored reference on non-acceptance, replayed over 1,800 GLM assignment positions, avoids 8 breakages but forfeits 6 repairs; no cell's accuracy-difference interval excludes zero (Appendix W). Assignment therefore does not make retention redundant. We cannot identify the corresponding replay for self-drafting arms whose overwritten incumbents were not retained, and preservation presupposes an available baseline artifact.

10 Conclusion

A net accuracy difference is two transitions and the paths that produced them. The audit reports both, and attributes each breakage to a refused call, an accepted wrong incumbent, or a failed rejection path. It needs no new mathematics, only an executor that records the path and the budget ledger next to the outcome.

Applied, it finds what a net number conceals: a loop whose effect spans zero while repairing $.244$ of failures and breaking $.138$ of successes, and provenance separating $5/347$ from $27/347$ breakages under budget parity. These workflows should be audited at the level of transitions and paths, not replaced by assignment, whose benefit is conditional.

AI USE STATEMENT

Generative AI tools were used to assist with experimental-design feedback, with literature discovery, with the implementation and debugging of the analysis and figure-generation code, with the creation and editing of scientific figures, with checking and interpreting results, and with drafting and editing parts of this manuscript. Numerical results were computed from the retained experimental records by executable scripts, not reported directly by an AI tool. The authors reviewed and tested all AI-assisted analysis code, citations, figures and manuscript text, and take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The paper and its appendices report the task splits and their sizes, the paired estimands, the model and verifier configurations, the composite execution budgets, the number of execution seeds per task, and the path-accounting rules that assign each transition to a terminal path. The anonymized supplementary material contains the workflow implementation, the reservation ledger, the preregistration, a checksum manifest identifying the frozen splits, and the offline analysis code together with derived per-task execution records: one row per task–seed pair carrying the grader's verdict, the budget ledger and the node trace from which acceptance, rejection and budget refusal are reconstructed. Generated text and task/request text in results, attempts, trajectories and rosters are replaced by SHA-256 digests and lengths. The path, signal-ablation, revert-control, verifier-fidelity and design-sensitivity analyses run on those records. Both parity studies and the serving outcome analysis include per-task inputs for offline recomputation; call records retain request/output digests, usage, timing and first-draft grading labels. The sampling replay uses frozen prefix-selection indices. Final symbolic regrades are preserved explicitly. Exact task membership and archive checksums are included. Raw-generation regrading and the original-cap code-parity ledger replay require inputs outside the archive. The underlying benchmarks are public: MBPP+ and HumanEval+ in the EvalPlus formulation, MATH, and BIG-Bench Hard.

REFERENCES

Berk Atıl, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Türe, et al. Non-determinism of "deterministic" llm system settings in hosted environments. In *Proceedings of the 5th Workshop on Evaluation and Comparison of NLP Systems*, pp. 135–148, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Deng Cai, Elman Mansimov, Yi-An Lai, Yixuan Su, Lei Shu, and Yi Zhang. Measuring and reducing model update regression in structured prediction for NLP. *Advances in Neural Information Processing Systems*, 35:19967–19979, 2022.

Hao Chen, Guanxi Lu, Yasuyuki Okoshi, Zhiwen Mo, Masato Motomura, and Hongxiang Fan. Rethinking optimal verification granularity for compute-efficient test-time scaling. *Advances in Neural Information Processing Systems*, 38:88650–88674, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chao Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on information theory*, 16(1):41–46, 1970.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Bradley Efron, Robert J Tibshirani, et al. *An introduction to the bootstrap*. Boca Raton, Florida, 2000.

Evrard Garcelon, Mohammad Ghavamzadeh, Alessandro Lazaric, and Matteo Pirotta. Conservative exploration in reinforcement learning. In *International conference on artificial intelligence and statistics*, pp. 1431–1441. PMLR, 2020.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *Advances in neural information processing systems*, 30, 2017.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

Yaojie Hu, Qiang Zhou, Qihong Chen, Xiaopeng Li, Linbo Liu, Dejiao Zhang, Amit Kachroo, Talha Oz, and Omer Tripp. Qualityflow: An agentic workflow for program synthesis controlled by llm quality checks. *arXiv preprint arXiv:2501.17167*, 2025b.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pp. 240–248. PMLR, 2016.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Changxin Ke, Rui Zhang, Jiaming Guo, Yuanbo Wen, Li Ding, Shuo Wang, Xuyuan Zhu, Xiong Peng, Di Huang, Zidong Du, et al. Qimeng-prepair: Precise code repair via edit-aware reward optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29114–29129, 2026.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 54523–54549, 2025.

Hareton KN Leung and Lee White. Insights into regression testing (software testing). In *Proceedings. Conference on Software Maintenance-1989*, pp. 60–69. IEEE, 1989.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of machine learning research*, 18(185):1–52, 2018.

Yang Li, Hai Liu, Dian Shao, Yu Wang, Xiyu Chen, Sergey Volkov, Bozhi Wang, Ziyu Sun, Sihang Liu, Ye Luo, et al. Agent-uct: Upper confidence bounds applied to trees for agentic workflow optimization with cost-awareness. *arXiv preprint arXiv:2607.24162*, 2026a.

Yushu Li, Wenlong Deng, Jiajin Li, and Xiaoxiao Li. Spend less, reason better: Budget-aware value tree search for llm agents. *arXiv preprint arXiv:2603.12634*, 2026b.

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.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

David Madras, Toni Pitassi, and Richard Zemel. Predict responsibly: improving fairness and accuracy by learning to defer. *Advances in neural information processing systems*, 31, 2018.

Hussein Mozannar and David Sontag. Consistent estimators for learning to defer to an expert. In *International conference on machine learning*, pp. 7076–7087. PMLR, 2020.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pp. I–I. Ieee, 2001.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Yuqing Xie, Yi-An Lai, Yuanjun Xiong, Yi Zhang, and Stefano Soatto. Regression bugs are in your model! measuring, reducing and analyzing regressions in NLP model updates. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 6589–6602, 2021.

Sijie Yan, Yuanjun Xiong, Kaustav Kundu, Shuo Yang, Siqi Deng, Meng Wang, Wei Xia, and Stefano Soatto. Positive-congruent training: Towards regression-free model updates. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14299–14308, 2021.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in llm inference. *Advances in Neural Information Processing Systems*, 38:169819–169851, 2026.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A NOTATION

Table 4: Symbols and event counts used across the tables; the middle column gives the conditioning population for rates and the counting population for counts. "Eligible" always means paired task–seed executions, clustered by task.

symbol	denominator	meaning
p	all pairs	baseline (stored-reference) accuracy
r	reference-wrong pairs	repair rate: workflow correct where the reference was wrong
b	reference-correct pairs	breakage rate: workflow wrong where the reference was correct
Δ	all pairs	net accuracy change, $(1-p)r - pb$
R	all pairs	executions whose first verifier decision was a rejection
R_B	reference-correct pairs	first rejections on a correct stored reference
E / G	first rejections	initial refinements executed / blocked by the ledger
VG	all pairs	executions blocked before the first verification
K_{pre}	reference-correct pairs	breakage before any verification
K_{acc}	reference-correct pairs	breakage on the acceptance path: the first verifier accepted a wrong incumbent
K_{rej}	reference-correct pairs	breakage after a first rejection, including budget-blocked fallbacks
λ	—	loss ratio: cost of one breakage in units of one repair
G_λ	all pairs	gross value $(1-p)r - \lambda pb$
λ^*	—	break-even loss ratio between two arms
C	held-out tasks	correct stored references in the certification split
Q	C	correct references the shared gate rejects
K_j	Q	observed breakages of candidate suffix j
U^{shared}, U_j^{stored}	C	shared pre-suffix ceiling and candidate-specific post-suffix bound
ICC	tasks	within-task correlation of the breakage indicator
ρ_{arm}	—	assumed cross-arm pairing structure in the design simulation

For task t with m_t matched seeds, let $\langle f \rangle = T^{-1} \sum_t m_t^{-1} \sum_{s=1}^{m_t} f_{ts}$, where $B_{ts}, W_{ts} \in \{0,1\}$ are correctness indicators. The complete-matrix estimators are $\hat{p} = \langle B \rangle$, $\hat{r} = \langle (1-B)W \rangle / \langle 1-B \rangle$ and $\hat{b} = \langle B(1-W) \rangle / \langle B \rangle$. Thus $\widehat{\Delta} = \langle W - B \rangle = (1-\hat{p})\hat{r} - \hat{p}\hat{b}$. A task without eligible seeds contributes zero to both the relevant numerator and denominator, not an undefined task-specific conditional rate. All tasks have three paired seeds in the main matrices, so these ratios equal pooled eligible-pair counts. The 383-position companion instead pools its available valid-reference pairs and resamples whole task clusters, retaining their available seeds. Its population and weighting are stated separately. U^{cert} denotes the held-out false-rejection upper bound in Appendix F, not the inspected-prefix U_j^{stored}.

B THE OPERATIONAL MATRIX IN THE REPAIR/BREAKAGE PLANE

C FINITE-SAMPLE RISK AUDIT

For C reference-correct examples and K observed breakages, the empirical rate is K/C. The pre-specified population certificate uses $(Q + \sqrt{n \log(1/\alpha)/2})/C$, clipped at one, where Q counts false rejections on n independent tasks. Conditional on the reference labels, Hoeffding bounds the sum of independent rejection indicators over the C correct references; replacing C by $n \geq C$ inside the square root is conservative. Its target is conditional false-rejection risk, which bounds breakage by Eq. 3. Exact binomial sensitivities require the additional i.i.d. Bernoulli model.

These bounds use independent tasks; the exact binomial sensitivity adds an i.i.d. Bernoulli model. The substitution of n for C is a property of the frozen rule. Using C directly gives a zero-rejection floor .04235 and an observed bound .06870 at $Q = 22, C = 835$, still above .05. The binomial bounds in Appendix F are post-hoc sensitivities; none replaces the frozen abstention. Variance-sensitive bounds would likewise need their assumptions and decision rule specified for a future audit.

For a fixed inspected prefix, let C be the number of correct stored references, Q the number rejected by the shared gate, and $K_j \leq Q$ the observed breakages of suffix j. For m contracts fixed before

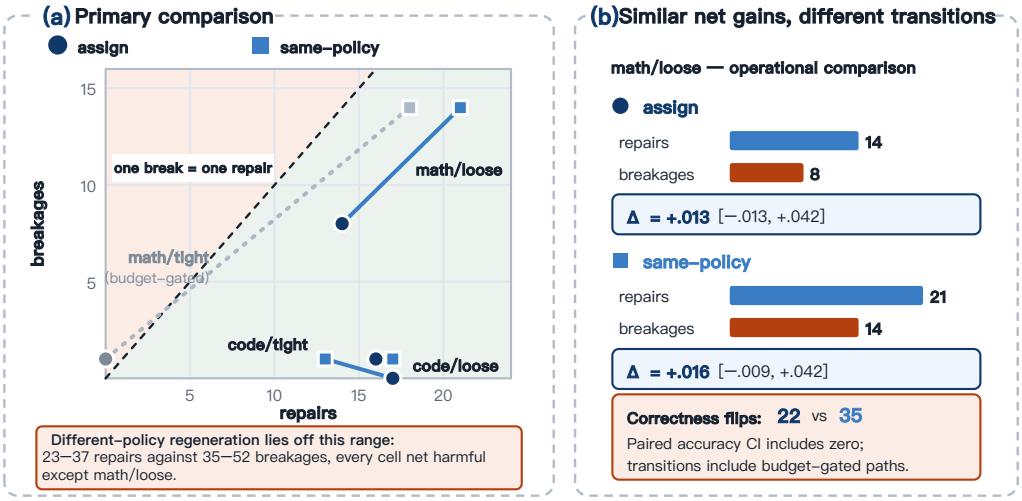

Figure 3: Net accuracy hides which transitions produced it, in the operational GLM-4-Flash matrix that motivated the budget-parity follow-up of §7. **(a)** Assignment against same-policy regeneration in the repair/breakage plane over 450 paired task–seed executions per arm; points below the dashed line raise gross value when a breakage costs as much as a repair. The math/`tight` pair is grey because its task-level cap blocks the suffix, so that contrast measures blocked execution rather than provenance. **(b)** The operational math/`loose` comparison: 22 correctness flips against 35, with a paired accuracy-difference interval including zero, and 19 same-policy initial refinements blocked against none for assignment.

their suffixes are inspected, and independent suffix draws conditional on that prefix, define

$$U^{\text{shared}} = \frac{Q}{C}, \qquad U_j^{\text{stored}} = \min\left\{\frac{Q}{C}, \frac{K_j + \sqrt{Q\log(m/\alpha)/2}}{C}\right\}. \tag{4}$$

The total-breakage interpretation requires a stored incumbent $I = B$, acceptance returning it unchanged, and no pre-verification breakage. Otherwise the expression controls only the inspected rejection suffix, not the regeneration workflow's total risk. The shared ceiling is available before any suffix executes, whereas the candidate-specific bound is available only afterward. Both condition on the particular tasks and shared-gate outcomes; neither is a future-task certificate.

D Clean-matrix diagnostics

The recorded solution preserves the original draft only when no refinement overwrites it. Consequently, all-draft byte identity is unavailable from these runs; Table 5 reports the exactly recoverable first-accepted estimand.

D.1 Content changes among first-accepted drafts

Table 7 separates first-verifier rejection from suffix execution. The math/`tight` comparison is censored: assignment reaches refinement in only $6/120$ rejected pairs, whereas same-policy regeneration reaches it in $0/14$. Those outcomes therefore cannot isolate incumbent provenance from suffix reachability.

Math/`loose` is an operational comparison: same-policy regeneration has 19 initial gates, whereas assignment has none. It jointly changes content source, the cost of an extra drafting call, and refinement access. The code assignment–same-policy contrasts have no initial gates; all profiles can stop later paths according to their configured fallbacks. Rejection rates use the 450 planned pairs, not only those reaching verification.

Table 5: Content changes among first-accepted same-policy drafts. Identical means exact byte equality; the four remaining columns partition byte-changing pairs by external-grader correctness. Both correct does not imply semantic or byte equivalence. These are acceptance-conditioned counts, not all-draft rates.

cell	accepted	identical	both correct	both wrong	repair	break
code/`tight`	355	337	13	2	2	1
code/`loose`	356	331	14	8	3	0
math/`tight`	86	28	54	4	0	0
math/`loose`	332	79	219	21	13	0

Table 6: The censored math/`tight` cell, excluded from Table 2 and from every provenance and loss conclusion. Columns match Table 2; this is the only cell where executions terminate before verification, so the K_{pre} term of Eq. 2 is non-empty.

cell	incumbent source	Δ [95% CI]	repair	break	accept	R/R_B	E/G
math/tight	assign	$-.002\,[-.007, +.000]$	0/103	1/347	0	120/47	6/114
	same-policy	$+.009\,[-.016, +.033]$	18/103	14/347	0	14/5	0/14
	different-policy	$-.007\,[-.049, +.038]$	35/103	38/347	1	8/4	0/8

D.2 Candidate-level verifier fidelity

Where the incumbent is assigned, its correctness is the stored reference's correctness, so the first verifier's decision can be scored against a known label without retaining rejected candidates. We treat "the incumbent is wrong" as the positive class and "the verifier rejects" as the positive prediction: precision is then the share of rejections that were warranted, and the false-positive rate the share of correct incumbents rejected, which is the quantity Eq. 3 bounds breakage by. All four cells verify every pair on this arm, so the censoring that excludes math/`tight` from the provenance conclusions does not affect its first-verifier counts. These are first-verification rates on this arm, not a model-independent verifier benchmark.

The two regimes separate. The visible-test verifier rejects a correct incumbent on $3/312$ and $3/314$ pairs, so the rejection-path exposure it creates is small and the bound of Eq. 3 is nearly tight. The gold-free self-check rejects a correct incumbent an order of magnitude more often, $47/347$ and $42/343$, which is why its rejection-path term carries most of the residual risk in mathematics. Neither verifier is strong at the other error: both accept a wrong incumbent on a fifth to a third of the pairs where one is present.

D.3 Parser and checker diagnostic

The 120 assignment rejections contain 19 stored answers without a parseable boxed answer, but none occurs among the 47 reference-correct rejections. One replay of the frozen gold-free checker for all 120 pairs yields 19 candidate parse failures, five checker parse failures, 58 numeric disagreements, 12 symbolic disagreements, and 26 agreements. Among the 47 reference-correct pairs these counts are 0, 3, 18, 4, and 22. The replay finishes normally on 111 and at the output cap on nine. A deterministic sample of 20 reference-correct rejections was manually checked: all incumbents have complete boxed answers; ten replays agree, eight disagree substantively, and two truncate before an answer. The checker prompt re-solves the problem without the incumbent text. These diagnostics do not support a formatting explanation for correct-reference rejection, but the original checker texts were not retained and the replay is not a historical reason attribution.

D.4 Widened-cap GLM companion

Widened-cap companion. A separate GLM companion removes blocking but does not fund drafting outside the suffix allowance, so it separates blocking from provenance only partially, unlike §7. Assignment and same-policy regeneration tie on *net* accuracy — each nets $+8$ over 383 eligible pairs, $16-8$ against $19-11$ — which is why their accuracy difference below is exactly $+.000$. The

Table 7: Control-flow outcomes in the clean matrix. VG is verification blocked by reservation; R_B is rejection with a correct stored reference; E/G are initial refinement executed/gated. K_{pre}, K_{acc}, and K_{rej} partition breakage before verification, after acceptance, and after rejection.

cell	source	VG	pass	reject	R_B	E	G	K_{pre}	K_{acc}	K_{rej}
code/tight	assign	0	353	97	3	97	0	0	0	0
code/tight	same-policy	0	355	95	3	95	0	0	1	0
code/tight	different-policy	0	313	137	59	82	55	0	22	28
code/loose	assign	0	355	95	3	95	0	0	0	1
code/loose	same-policy	0	356	94	6	94	0	0	0	1
code/loose	different-policy	0	314	136	62	136	0	0	30	22
math/tight	assign	0	330	120	47	6	114	0	0	1
math/tight	same-policy	350	86	14	5	0	14	14	0	0
math/tight	different-policy	364	78	8	4	0	8	35	1	2
math/loose	assign	0	325	125	42	125	0	0	0	8
math/loose	same-policy	0	332	118	47	99	19	0	0	14
math/loose	different-policy	0	307	143	66	126	17	0	0	35

Table 8: Measured clean-matrix cost by source. Reference acquisition is separate from the intervention cost and is sunk when those outputs already exist. Tokens include inputs and outputs.

cell	source	calls	tokens
code/tight	reference	450	60,744
	assign	304	107,015
	same-policy	690	144,356
	different-policy	594	279,580
code/loose	reference	450	60,620
	assign	307	113,826
	same-policy	755	169,803
	different-policy	840	426,606
math/tight	reference	450	359,652
	assign	456	364,975
	same-policy	550	409,352
	different-policy	536	409,236
math/loose	reference	450	358,359
	assign	732	757,624
	same-policy	1,073	956,646
	different-policy	1,120	1,026,340

tie is in the net only: the two arms break 8 and 11 of 294 correct references. The tables below give both.

This math/`tight` companion retains the drafting, checker, and suffix contracts and per-call output limits, while allowing nine LLM calls, 128,000 input tokens, and 13,824 output tokens per task. The analysis uses the same 383 task–seed positions with completed stored references in all three arms: 294 correct and 89 incorrect references. Table 10 describes this valid-reference subset, not all-task accuracy. Every initial rejection reaches refinement; later fallbacks remain part of the workflow outcome. Selection depends only on pretreatment reference execution status. The reference cohort is the earlier `glm4flash_repaired_code_20260909` run, distinct from Appendix P's completed-reference study. The 67 excluded references comprise 64 rate-limit errors, two connection errors and one timeout. All three companion arms complete their 450 recorded positions, and selection is not based on their outcomes.

Direct assignment–same-policy differences pool the eligible pairs and resample task clusters, retaining each task's available eligible seeds. Repair and breakage differences condition on incorrect and correct references, respectively; accuracy and gross value use all 383 eligible pairs. The 95%

Table 9: First-verifier fidelity on the assignment arm, where the incumbent's label is the stored reference's. Positive class is a wrong incumbent; positive prediction is rejection. TP/FP/FN/TN are pair counts over each cell's 450 matched task–seed pairs.

cell	verifier	TP	FP	FN	TN	precision	FPR	FNR	MCC
code/`tight`	visible tests	94	3	44	309	.969	.010	.319	.753
code/`loose`	visible tests	92	3	44	311	.968	.010	.324	.750
math/`tight`	gold-free self-check	73	47	30	300	.608	.135	.291	.545
math/`loose`	gold-free self-check	83	42	24	301	.664	.122	.224	.621

Table 10: GLM widened-cap companion on valid stored references. Denominators are 89 for repairs and 294 for breakages. Definitions match Table 2.

source	repairs	breakages	accept	R/R_B	E/G
Assignment	16	8	0	98/30	98/0
Same-policy	19	11	3	92/33	92/0
Different-policy	29	31	5	110/55	110/0

intervals use 10,000 paired bootstrap resamples and are descriptive, unadjusted, and conditional on the valid-reference subset.

Table 11: Widened-cap GLM: assignment minus same-policy, on valid references.

repair-rate difference	breakage-rate difference	accuracy difference
$-.034$ [-.141, +.067]	$-.010$ [-.035, +.013]	$+.000$ [-.031, +.031]

Same-policy regeneration incurs three acceptance-path breakages here despite zero blocking, against none for assignment, and they come from two tasks rather than three independent failures. On a divisibility problem the correct reference counts 11 numbers of the form $25AB$ while the regenerated draft and the independent checker both answer three, omitting the digit-sum case $A + B = 11$; this recurs at two seeds. On a candy-transfer problem a draft and checker agree on four rather than the correct five statements. Agreement can therefore accept a wrong replacement with no refinement at all. The cases demonstrate that the path exists without establishing a frequent or resolved advantage for assignment.

For assignment, FRR is $30/294 = .1020$ while breakage is $8/294 = .0272$; the difference is the slack in Eq. 3. In the primary math/`tight` cell, $47/347 = .1354$ is an operational false-rejection rate under different path access. Neither rate can be substituted for the held-out study's $22/835 = .0263$ calibration rate: they condition on different reference samples and execution regimes.

E Direct assignment–regeneration contrasts

Preservation has a price. It forgoes repairs, so it pays only above a break-even loss ratio — 1.17 against same-policy in math/`loose`, 1.65 against different-policy in the serial control — and paired gross-value intervals include zero in every cell of this matrix; the break-even ratios are sample point estimates, not confidence-guaranteed decision thresholds (Appendix E).

Table 13 reports direct paired differences rather than comparing two marginal intervals. Resampling is over the 150 task clusters and retains the three execution seeds within each task. The math/`tight` row is excluded from content-provenance and loss conclusions; math/`loose` describes full workflows with differential budget access, not a gate-free content intervention.

F Held-out decision and loss sensitivity

The frozen candidate was evaluated on 1,194 disjoint MATH tasks. Among 835 reference-correct tasks, the verifier falsely rejected 22 and the workflow broke two. The prespecified one-sided Hoeffding FRR bound is .07699, above $\epsilon = .05$, so the deployment decision is *abstain*. Exact binomial

Table 12: Widened-cap GLM: assignment minus same-policy gross value per eligible pair, before inference cost.

$\lambda = 1$	$\lambda = 3$	$\lambda = 5$
+.000 [-.031, +.031]	+.016 [-.044, +.078]	+.031 [-.062, +.130]

Table 13: Assignment minus same-policy regeneration, with task-clustered 95% bootstrap intervals.

cell	repair-rate difference	breakage-rate difference	accuracy difference
code/`tight`	$+0.029\,[-0.024, +0.088]$	$-0.003\,[-0.010, +0.000]$	$+0.011\,[-0.004, +0.029]$
code/`loose`	$-0.007\,[-0.051, +0.038]$	$+0.000\,[-0.009, +0.009]$	$-0.002\,[-0.016, +0.011]$
math/`tight`	$-0.175\,[-0.263, -0.101]$	$-0.037\,[-0.065, -0.014]$	$-0.011\,[-0.036, +0.013]$
math/`loose`	$-0.065\,[-0.156, +0.024]$	$-0.017\,[-0.047, +0.013]$	$-0.002\,[-0.031, +0.027]$

sensitivity bounds are .03741 for FRR and .00752 for breakage; they do not replace the prespecified decision. Overall accuracy changes by $+.0034\,[-.0008, .0084]$. Six repairs and two breakages yield gross value $(6 - 2\lambda)/1194$, positive only for $\lambda < 3$ before inference cost.

In the three main operational cells, assignment versus same-policy regeneration has the following loss interpretation: code/`tight` adds four repairs and avoids one break; code/`loose` gives up one repair without avoiding a break; and math/`loose` gives up seven repairs while avoiding six breaks, so its sample gross-value difference is positive only for $\lambda > 7/6 \approx 1.17$; the reported intervals do not establish positive population value at those ratios.

G Workflow-search operator audit

Table 15 records the narrow interface claim used in the main text. "No" means that the primary paper and documented official-repository interface expose no named provenance-preserving assignment primitive; it does not imply that custom code cannot express the behavior.

Exploratory search intervention. One code-search seed adds a stored-assignment initializer and a generator-to-assignment mutation while retaining the search loop and its accuracy–cost objective. Of six evaluated candidates, an assignment-containing candidate enters the archive and ranks first by mean objective. This shows that the primitive is executable and reachable by the search, not that it improves search performance across seeds or unseen tasks. We do not use this one-seed result as evidence for a default workflow choice.

H Local Qwen serial-serving control

The serial control summarised in §7 uses Qwen2.5-Coder-7B-Instruct with fixed weights, served in BF16 by vLLM 0.29.0 on one NVIDIA RTX PRO 6000 Blackwell GPU. The server admits one sequence at a time, disables prefix caching, and forwards the task seed. Same-policy drafting uses temperature zero; the different-policy stress test uses temperature 0.7. Response caching is disabled in every arm.

The 150-task MATH split and all workflow code, prompts, verifier logic, suffix, per-call output limits, and seeds are shared with the provenance intervention. The task-level envelope is widened to nine calls, 128,000 input tokens, and 13,824 output tokens so reservation admission does not block initial refinement. Each source covers the same 150 tasks and three seeds, giving 450 reference rows and 450 rows per intervention, with every planned paired position present. Of the 1,800 rows, 1,786 have completed status and 14 have the later token-settlement overruns described below.

The byte-identity audit conditions only on same-policy drafts accepted at the first verifier, where the independently generated draft remains observable without subsequent refinement. All $306/306$ such drafts are byte-identical to the stored reference. The drivers use the same prompt, model revision, decoding configuration, seed, and server; complete request bodies were not archived, so the attempt logs alone do not establish full-payload equality. Assignment and same-policy regeneration also

Table 14: Assignment minus same-policy gross value per input, $(\mathrm{Rep}_A - \mathrm{Rep}_S - \lambda(K_A - K_S))/450$, at illustrative loss ratios. Direct paired 95% intervals resample 150 task clusters, retain all three seeds, exclude inference cost, and are unadjusted across ratios. Math/`loose` includes its operational budget fallbacks.

cell	$\lambda = 1$	$\lambda = 3$	$\lambda = 5$
code/tight	$+0.011\,[-0.004, +0.029]$	$+0.016\,[-0.004, +0.038]$	$+0.020\,[-0.004, +0.049]$
code/loose	$-0.002\,[-0.016, +0.011]$	$-0.002\,[-0.022, +0.018]$	$-0.002\,[-0.033, +0.027]$
math/loose	$-0.002\,[-0.031, +0.027]$	$+0.024\,[-0.047, +0.093]$	$+0.051\,[-0.064, +0.162]$

Table 15: Primary-paper and official-repository audit of documented search interfaces.

System	Documented search units or moves	Stored-reference assignment
GPTSwarm	operation nodes; prompt and inter-agent edge optimization	No
ADAS	archive-guided generation of arbitrary agent code	Expressible, not first-class
AFlow	Generate, Format, Review, Revise, Ensemble, Test, Programmer; MCTS code edits	No
EvoFlow	tagged parent retrieval, crossover, mutation, niching selection	No
AgentSquare	Planning, Reasoning, Tool Use, Memory; module evolution/recombination	No

have identical aggregate repair, breakage, accuracy, and rejection counts (Table 16). This is evidence for equivalence under the tested serving contract, not a claim that nominal temperature zero alone guarantees reproducibility.

Table 16: Local Qwen math/`tight` control: 150 tasks and three seeds per source. R is first rejection and R_B rejection of a correct reference; E/G denotes initial refinement executed/blocked. Reference-correct and reference-wrong denominators are 309 and 141.

source	repairs	breakages	R	R_B	E/G
Assignment	17	6	144	31	144/0
Same-policy	17	6	144	31	144/0
Different-policy	50	26	141	50	141/0

The 14 later input-token settlement overruns occur at the same seven task–seed positions in assignment and same-policy regeneration. Input-token reservation uses a cross-tokenizer proxy and fixed margin; actual usage exceeds that reservation on these long prompts and interrupts the trajectory after a response. These are not initial reservation gates: all 429 first-verifier rejections across the three arms enter the first refinement. This control therefore distinguishes initial admission from later execution failure, rather than claiming every refinement trajectory completed.

I Conditional five-suffix audit

Table 17 gives the disaggregated outcomes behind the inspected common prefix evaluated by the conditional bound in Eq. 4. These are measured marginal suffix costs and paired transitions on that fixed prefix; they are not an independent population certificate or a deployment-utility comparison.

Length-terminated trajectories account for $3/4$, $6/8$, $14/14$, $1/1$, and $7/9$ breakages in the standard, targeted, conservative, long-form, and independent-reconcile contracts, respectively: $31/36$ overall. Here length means at least one suffix response hits its output limit; for the two-call reconciliation contract this need not be the final response. The descriptive accuracy changes for the long-form and conservative contracts are $+.0067$ $[+.0017, +.0126]$ and $-.0084$ $[-.0151, -.0017]$, respectively. These unadjusted intervals come from the inspected prefix and are not an independent positive deployment validation. The contrast jointly changes prompt, temperature, and output limits and cannot identify their separate effects.

Table 17: Five fresh suffixes behind one inspected common prefix. Costs are marginal suffix totals; length counts trajectories with at least one response terminated at the output cap. $U_j = U_j^{\text{stored}}$ is the simultaneous conditional stored-prefix bound in Eq. 4, not a future-task certificate. The inspected population has $N = 1,194$, $C = 835$ correct references and $Q = 22$ false rejections; $m = 5$ suffixes and $\alpha = .05$. Accuracy changes use all N tasks, not the suffix-call count.

refiner	calls	tokens	repair	break	length	U_j
standard	100	251,954	6	4	48	.01331
targeted	100	169,273	7	8	54	.01810
conservative	100	194,325	4	14	69	.02529
long-form	100	276,325	9	1	30	.00972
independent+reconcile	200	341,316	6	9	61	.01930

Five frozen refiner contracts behind one stored reference and gate show that a common gate does not make suffix risk uniform: repairs range from four to nine and breakages from one to fourteen, and 31 of the 36 breakages occur on trajectories containing a length-terminated response, so these executions do not establish that a conservative prompt preserves answers under this output cap. Length termination is an observed association; its causal contribution and behavior under a larger cap were not tested.

J BUDGET PROFILES AND THE INTERMEDIATE ENVELOPE

Table 1 omits the intermediate `unseen` profile, which was executed under the same freeze and interpolates as the ordering predicts. The profiles are `tight` = (4 calls, 8k in, 2k out, $0.10), `unseen` = (6, 12k, 3k, $0.15) and `loose` = (8, 16k, 4k, $0.25).

Table 18: Intermediate preregistered budget profile, with the same estimands and task-clustered intervals as Table 1.

Family	Budget	Workflow	Δ (95% CI)	repair	breakage	$/task
code	unseen	vanilla	-0.038 $[-0.093, +0.018]$	0.252	0.147	0.0085
code	unseen	protected	$+0.020$ $[+0.002, +0.042]$	0.073	0.000	0.0025
math	unseen	vanilla	$+0.013$ $[-0.020, +0.044]$	0.233	0.067	0.0141
math	unseen	protected	$+0.000$ $[-0.009, +0.009]$	0.017	0.006	0.0141

Figure 4 gives the full budget-profile sweep for both families.

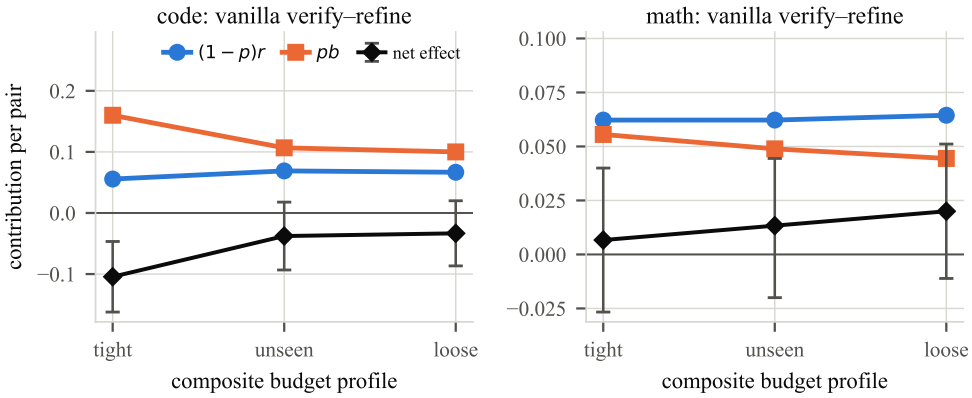

Figure 4: The full budget-profile sweep. The left panel is the code family reproduced from Figure 2; the right panel is mathematics, where the weighted contributions change less across profiles. Both panels use the unconditional contributions $(1 - p)r$ and pb. Frozen test splits, 95% CIs on the net effect.

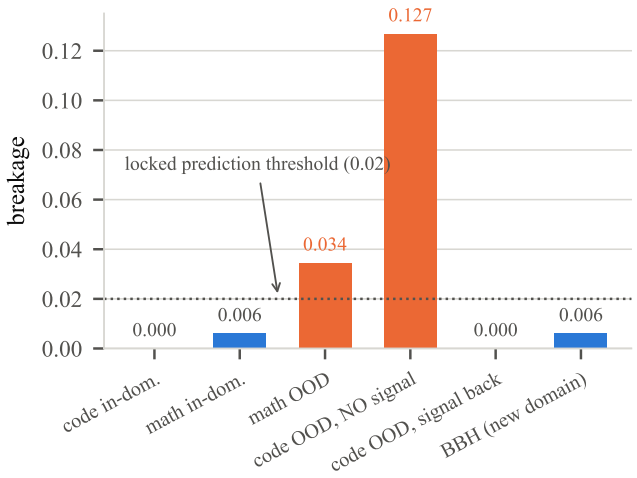

Figure 5: Protected-arm breakage in selected GPT-4o conditions, including prospectively evaluated BIG-Bench Hard. Mathematics OOD is an exception to the 0.02 threshold despite verifier signal (0.034). Code OOD without tests reaches 0.127; restoring tests on the matched 68-task subset returns breakage to zero. Signal does not guarantee breakage below the threshold.

Figure 5 compares selected GPT-4o conditions. On 120 held-out BIG-Bench Hard tasks (Suzgun et al., 2023), protected breakage is 0–0.006 with non-negative net estimates. Restoring tests on the matched 68-task subset returns protected breakage to zero.

K SENSITIVITY TO THE LEDGER DEFECT

4.7% of frozen-test executions (466/9,900) terminated with a settlement overrun, unevenly across arms (protected 6.5%, vanilla 4.5%, baseline 3.6%), and were scored as failures. We recompute every headline contrast on the subset of tasks where neither arm hit the defect in any seed.

Contrast	all tasks	defect-free subset	n
C1 code loose, vanilla − direct	-0.033 $[-0.087, +0.020]$	-0.046 $[-0.096, +0.005]$	146
C2 code tight, vanilla − direct	-0.104 $[-0.162, -0.047]$	-0.115 $[-0.173, -0.056]$	148
C3 code loose, no signal	-0.118 $[-0.178, -0.058]$	-0.132 $[-0.192, -0.074]$	144
C4 code tight, protected	$+0.013$ $[+0.002, +0.031]$	$+0.014$ $[+0.002, +0.032]$	147
C4 code loose, protected	$+0.020$ $[+0.002, +0.042]$	$+0.018$ $[+0.002, +0.041]$	145
C4 code unseen, protected	$+0.020$ $[+0.002, +0.042]$	$+0.021$ $[+0.002, +0.043]$	146

Table 19: Mathematics ledger-defect sensitivity, with final symbolic grading. Each comparison retains only tasks with no settlement overrun in either arm at any seed. Both columns are recomputed with 10,000 paired task-cluster resamples; n is the retained task count. These post-hoc conditional subsets are not randomized interventions on the defect.

contrast	all tasks	no-overrun subset	n
math/`tight`, vanilla	+0.007 [-0.027, +0.040]	+0.007 [-0.029, +0.043]	139
math/`tight`, protected	+0.000 [+0.000, +0.000]	+0.000 [+0.000, +0.000]	139
math/`loose`, vanilla	+0.020 [-0.011, +0.051]	+0.036 [+0.009, +0.067]	110
math/`loose`, protected	+0.007 [-0.002, +0.018]	+0.003 [-0.006, +0.012]	114

Mathematics is more sensitive to this restriction: vanilla at `loose` changes from $+.020$ $[-.011, +.051]$ on all tasks to $+.036$ $[+.009, +.067]$ on 110 retained tasks. The other three mathematics intervals include zero. Thus the restriction can change whether a mathematics interval excludes zero; it is not a claim that every headline inference is unchanged. All-task results remain primary.

Breakage of the protected construction is 0.000 under both views in the code cells reported here. The directions of the code estimates persist in the no-overrun subsets; this conditional analysis does not identify the causal effect of the ledger defect.

L AUTOMATED SEARCH: PROTOCOL AND PER-CLAIM RESULTS

This appendix is an exploratory audit of our own design bias, not a result. Incumbent protection was derived by us, from our own decomposition, and tested by us. We therefore asked whether a search procedure, given a design space we specified in advance and no access to that derivation, selects the same construction. It is a bound on our design bias, not an independent replication: the node whitelist, the edge grammar and the objective are ours.

The searcher evolves workflows in a restricted language: at most eight nodes, at most two loops of three iterations, a fixed node whitelist, no model-generated executable code, and static validation before any spend. Multi-fidelity racing uses nested task subsets (Jamieson & Talwalkar, 2016); candidates are scored jointly at both searched budgets by an objective that averages per-budget accuracy with the worst budget. The budget-contingent and static arms differ only in whether edges may read remaining budget. Across 200 sampled workflows the static arm never emitted a budget predicate, and Protocol A gives it a full search budget per tier, twice the budget of the budget-contingent arm. At full fidelity on the search split every arm selected the same shape: a low-temperature draft, a verifier gate, refinement only on rejection. This is not the search rediscovering our principle. Every candidate in the space carries the verifier gate, so selecting it is not evidence about the mechanism of equation 2; and the space does not contain the choice that matters, namely whether to reuse a reference system's stored output as the incumbent. What the search does show is a preference for low-temperature drafts, which is consistent with, but does not isolate, reference preservation. Of five preregistered confirmatory claims (`prereg-freeze-partII`), three fail: non-inferiority holds in code at both budgets and at math/`loose` but not at math/`tight`; the cost reduction reaches 10.5% against a 20% threshold, and in math the searched policy is 191% more expensive in exchange for $+0.049$ accuracy, a Pareto trade rather than a saving; and only 3 of 12 selected policies retain the gated shape, because the preregistered one-shot selection uses a 40-task validation split whose variance swamps the 1–2 point gaps between candidates. What survives is narrow: at an unseen \$0.15 budget the code policy is indistinguishable from the preregistered static control (-0.001 $[-0.013, +0.010]$) at half the search cost, and a search over our design space does select the gated pattern there. A larger validation split would be needed to resolve the small candidate gaps; we did not retrofit it after the freeze.

Verifier fidelity. Table 9 extends the coarse signal taxonomy with assignment-arm precision, FPR and FNR; these measurements do not cover every regenerated candidate or serving regime.

M OFFLINE SAMPLING REPLAYS

These analyses use only data already on disk. Each frozen-test task was drafted three times for the seed-averaged baseline. Math uses majority voting over normalized answers (ties by seed); code selects the lowest-indexed draft passing visible tests and falls back to seed 0. Because three direct calls cost 1.3–2$\times$ the protected arm, we also interpolate between adjacent prefixes using weights fixed from logged cost alone: code uses $k = 2$ on 54.4% of tasks and $k = 1$ otherwise; math uses $k = 3$ on 23.3% and $k = 2$ otherwise. This matches expected mean dollars, not each task's realized path, and every output comes from one dependent seed pool.

Against the matched mixture, the task-paired bootstrap difference is $+0.019$ $[-0.001, +0.044]$ in code and $+0.007$ $[-0.018, +0.031]$ in math. Neither interval excludes zero. These are dependent-pool sensitivity results. `python scripts/matched_cost_replay.py` recomputes the comparison from frozen selection indices. Refinement-depth stratification is descriptive because earlier verifier decisions select depth.

Table 20: Reference preservation against offline sampling replays, code and math loose. The matched row is an expected mixture of adjacent prefixes.

	baseline p	accuracy	\$/task
code, visible-test selection ($k = 3$)	0.727	0.733	0.00489
code, cost-matched sampling mixture	0.727	0.727	0.00251
code, reference-preserving	0.727	0.747	0.00251
math, self-consistency vote ($k = 3$)	0.733	0.753	0.02057
math, cost-matched sampling mixture	0.733	0.733	0.01532
math, reference-preserving	0.733	0.740	0.01532

Table 21: What each contrast resolves. Left: assignment minus same-policy regeneration, task-clustered 95% intervals. Where an arm has no observed breakage the percentile bootstrap cannot produce harm it never saw, so that endpoint is degenerate and excludes nothing (§8). Right: clusters contributing at least one reference-correct seed and the eligible positions they hold, the same-policy breakage rate and its intraclass correlation, and the simulated power and screened-task count for a prespecified halving of breakage. Power and task counts are spans over cross-arm pairing structures the design does not identify, and are design-sensitivity calculations conditional on plug-in estimates of the rate, correlation and eligibility profile, not observed power.

	assignment − same-policy, 95% CI			design resolution			
cell	repair	breakage	accuracy	clust./pos.	rate / ICC	power	$n_{80\%}$
code/`tight`	+.029 [-.023, +.090]	−.003 [-.010, +.000]	+.011 [-.004, +.031]	106/312	.0032 / .00	.00–.01	2,400–7,100
code/`loose`	−.007 [-.050, +.037]	+.000 [-.010, +.009]	−.002 [-.016, +.011]	106/314	.0032 / .25	.00–.00	2,550–6,000
math/`loose`	−.065 [-.156, +.025]	−.017 [-.048, +.013]	−.002 [-.031, +.027]	124/343	.0408 / .39	.36–.66	200–450

N EFFECT PRECISION AND DESIGN SURFACE

The operational matrix's same-policy contrasts all include zero, while §7 resolves a large one. Both follow from one design fact: a 150-task cell detects a four-to-fivefold difference in breakage and not a moderate one. What matters is the denominator. Breakage is defined only where the stored reference is correct, so a cell's 450 paired executions supply 312–343 eligible positions in 106–124 contributing clusters, 25% of screened tasks contribute none, and within a cluster the breakage indicator has intraclass correlation .00, .25 or .39. Treating the design as 450 independent trials overstates its precision severalfold.

The intervals here and in Table 13 use separate 10,000-draw task bootstraps (seeds 20260916 and 20260915); small last-digit differences reflect Monte Carlo variation on the same records.

Tolerable differences. In math/`loose`, where both arms break, the data tolerate assignment giving up as much as 15.6 repair points while avoiding as many as 4.8, and accuracy differences within ±3.1 points. No cell here serves a practitioner who would switch on a one-point breakage reduction. The code breakage intervals must be read differently. Assignment records no breakage at all in code/`tight`, so a percentile bootstrap cannot resample harm it never observed: that endpoint is pinned at zero by construction and excludes nothing about population risk. For the code cells we therefore report the sample counts, $0/312$ against $1/312$ and $1/314$ against $1/314$, and treat the direction as unresolved.

Detectability, not observed power. The right-hand block of Table 21 simulates the executed design under prespecified effects. The effect sizes are prespecified; the event rate, ICC and eligibility profile are plug-in estimates from the executed cell. No entry reports observed power. Because the design does not identify how the arms' breakages are paired, we simulate across paired structures and report the span. At the code cells' rate of .0032 the design detects a halving with probability at most .01 under every structure; at math/`loose`'s .0408 it reaches .36–.66. The code nulls therefore carry essentially no evidence against a moderate effect, and neither is an equivalence result.

There is no single minimum detectable effect. Detectability depends jointly on the event rate, the within-task correlation, the eligible denominator and the paired structure, and the last is not identified by an executed audit. The task counts in Table 21 are scenarios, not requirements: 80% power against a halving of code breakage needs 2,400 screened tasks under the most favourable paired structure and 7,100 under the least; Table 22 gives the full surface.

Table 22 reports the simulated detectability surface behind §8. Each row draws a task-level breakage propensity with mean equal to the stated rate and the stated intraclass correlation, draws three exchangeable seed outcomes per task in each arm, and applies a paired test over the task-level mean differences, so the unit of replication is always the task. The prespecified effect is a proportional reduction of the breakage rate in the protected arm. Power at the null is reported as a calibration check: it is at nominal level except at the sparsest high-correlation settings, where the test is conservative and the tabulated power is correspondingly understated. The surface below uses a prespecified grid of event rates and correlations, whereas the per-cell figures in Table 21 condition on plug-in estimates of each cell's event rate, within-task correlation and eligibility profile. Both fix the effect size in advance, and neither is an observed-power calculation. Each entry uses 4,000 simulations.

Table 22: Detectability of a prespecified proportional halving of breakage. Rows vary the *within-task* intraclass correlation; the three column groups vary ρ_{arm}, the *cross-arm* pairing structure, which the executed design does not identify. MDE is the smallest proportional reduction reaching 80% power; "–" means no reduction up to complete elimination reaches it. Task counts are screened tasks at three seeds each.

			$\rho_{\mathrm{arm}} = 0$		$\rho_{\mathrm{arm}} = 0.5$		$\rho_{\mathrm{arm}} = 0.9$	
rate	ICC	tasks	power	MDE	power	MDE	power	MDE
.005	.0	150	.03	–	.03	–	.01	–
.005	.0	300	.09	–	.10	–	.09	–
.005	.0	600	.16	1.00	.22	.95	.36	.95
.005	.0	1,200	.29	.85	.42	.75	.69	.60
.005	.0	2,400	.54	.65	.73	.55	.95	.40
.005	.5	150	.00	–	.00	–	.00	–
.005	.5	300	.02	–	.02	–	.02	–
.005	.5	600	.11	–	.13	–	.15	–
.005	.5	1,200	.33	.90	.44	.85	.55	.75
.005	.5	2,400	.67	.60	.80	.55	.94	.40
.020	.0	150	.16	1.00	.22	1.00	.33	.95
.020	.0	300	.31	.85	.42	.80	.69	.60
.020	.0	600	.52	.65	.74	.55	.96	.40
.020	.0	1,200	.83	.50	.95	.40	1.00	.25
.020	.0	2,400	.98	.35	.99	.30	1.00	.20
.020	.5	150	.12	–	.13	–	.14	–
.020	.5	300	.34	.90	.42	.85	.56	.80
.020	.5	600	.67	.60	.80	.55	.94	.40
.020	.5	1,200	.93	.45	.98	.35	.99	.20
.020	.5	2,400	.99	.30	1.00	.25	1.00	.15
.080	.0	150	.56	.65	.74	.55	.95	.40
.080	.0	300	.85	.50	.96	.40	.99	.25
.080	.0	600	.99	.35	.99	.30	1.00	.20
.080	.0	1,200	1.00	.30	1.00	.20	1.00	.15
.080	.0	2,400	1.00	.20	1.00	.15	1.00	.10
.080	.5	150	.69	.60	.82	.50	.94	.40
.080	.5	300	.95	.40	.98	.35	.99	.25
.080	.5	600	.99	.30	1.00	.25	1.00	.15
.080	.5	1,200	1.00	.20	1.00	.15	1.00	.10
.080	.5	2,400	1.00	.15	1.00	.15	1.00	.10

Table 23: Null calibration for the full frozen simulation grid, 4,000 simulations per entry at nominal two-sided $\alpha = .05$. Columns give the empirical rejection probability for the paired test at zero breakage reduction; ρ_{arm} is the assumed cross-arm dependence. The Monte Carlo standard error near .05 is approximately .0034.

rate	within-task ICC	screened tasks	$\rho_{\mathrm{arm}} = 0$	.5	.9
0.005	0.0	150	0.0272	0.0073	0.0000
0.005	0.0	300	0.0400	0.0275	0.0003

Table 23 continued

rate	within-task ICC	screened tasks	$\rho_{\text{arm}} = 0$	.5	.9
0.005	0.0	600	0.0452	0.0393	0.0037
0.005	0.0	1,200	0.0500	0.0437	0.0210
0.005	0.0	2,400	0.0498	0.0470	0.0450
0.005	0.2	150	0.0050	0.0020	0.0000
0.005	0.2	300	0.0260	0.0130	0.0003
0.005	0.2	600	0.0405	0.0372	0.0022
0.005	0.2	1,200	0.0510	0.0420	0.0118
0.005	0.2	2,400	0.0493	0.0542	0.0343
0.005	0.5	150	0.0005	0.0003	0.0000
0.005	0.5	300	0.0050	0.0030	0.0000
0.005	0.5	600	0.0320	0.0150	0.0003
0.005	0.5	1,200	0.0418	0.0370	0.0030
0.005	0.5	2,400	0.0437	0.0470	0.0220
0.010	0.0	150	0.0457	0.0338	0.0008
0.010	0.0	300	0.0415	0.0423	0.0030
0.010	0.0	600	0.0522	0.0510	0.0205
0.010	0.0	1,200	0.0535	0.0465	0.0450
0.010	0.0	2,400	0.0470	0.0488	0.0423
0.010	0.2	150	0.0297	0.0127	0.0003
0.010	0.2	300	0.0428	0.0362	0.0027
0.010	0.2	600	0.0493	0.0385	0.0127
0.010	0.2	1,200	0.0428	0.0462	0.0370
0.010	0.2	2,400	0.0393	0.0512	0.0440
0.010	0.5	150	0.0065	0.0037	0.0000
0.010	0.5	300	0.0253	0.0147	0.0000
0.010	0.5	600	0.0420	0.0405	0.0022
0.010	0.5	1,200	0.0512	0.0462	0.0182
0.010	0.5	2,400	0.0455	0.0520	0.0430
0.020	0.0	150	0.0455	0.0450	0.0050
0.020	0.0	300	0.0550	0.0420	0.0217
0.020	0.0	600	0.0530	0.0450	0.0470
0.020	0.0	1,200	0.0542	0.0462	0.0408
0.020	0.0	2,400	0.0525	0.0505	0.0480
0.020	0.2	150	0.0398	0.0343	0.0015
0.020	0.2	300	0.0485	0.0435	0.0123
0.020	0.2	600	0.0467	0.0452	0.0348
0.020	0.2	1,200	0.0490	0.0485	0.0432
0.020	0.2	2,400	0.0550	0.0532	0.0457
0.020	0.5	150	0.0230	0.0125	0.0008
0.020	0.5	300	0.0398	0.0350	0.0037
0.020	0.5	600	0.0503	0.0398	0.0190
0.020	0.5	1,200	0.0503	0.0498	0.0408
0.020	0.5	2,400	0.0493	0.0418	0.0415
0.040	0.0	150	0.0440	0.0488	0.0215
0.040	0.0	300	0.0465	0.0530	0.0430
0.040	0.0	600	0.0485	0.0553	0.0450
0.040	0.0	1,200	0.0525	0.0512	0.0508
0.040	0.0	2,400	0.0475	0.0498	0.0460
0.040	0.2	150	0.0450	0.0415	0.0132
0.040	0.2	300	0.0473	0.0510	0.0333
0.040	0.2	600	0.0540	0.0515	0.0437
0.040	0.2	1,200	0.0428	0.0460	0.0488
0.040	0.2	2,400	0.0467	0.0485	0.0500
0.040	0.5	150	0.0420	0.0325	0.0030
0.040	0.5	300	0.0493	0.0478	0.0187
0.040	0.5	600	0.0525	0.0462	0.0387

Table 23 continued

rate	within-task ICC	screened tasks	$\rho_{\text{arm}} = 0$	.5	.9
0.040	0.5	1,200	0.0508	0.0565	0.0428
0.040	0.5	2,400	0.0508	0.0545	0.0500
0.080	0.0	150	0.0532	0.0550	0.0387
0.080	0.0	300	0.0510	0.0508	0.0442
0.080	0.0	600	0.0473	0.0498	0.0503
0.080	0.0	1,200	0.0488	0.0457	0.0525
0.080	0.0	2,400	0.0483	0.0500	0.0535
0.080	0.2	150	0.0465	0.0498	0.0408
0.080	0.2	300	0.0490	0.0470	0.0473
0.080	0.2	600	0.0495	0.0490	0.0522
0.080	0.2	1,200	0.0535	0.0500	0.0558
0.080	0.2	2,400	0.0520	0.0583	0.0495
0.080	0.5	150	0.0515	0.0510	0.0173
0.080	0.5	300	0.0462	0.0488	0.0375
0.080	0.5	600	0.0548	0.0522	0.0442
0.080	0.5	1,200	0.0545	0.0475	0.0488
0.080	0.5	2,400	0.0517	0.0457	0.0505

The frozen grid's null rejection probabilities range from 0 to $.05825$. The very sparse settings can be strongly conservative; these diagnostics calibrate the stated simulator and test, not every possible data-generating process. The full grid is included to make the calibration claim inspectable.

O KIMI K3 REPLICATION

The preregistered-design study of §5 uses GPT-4o. As a check on whether the qualitative pattern, with breakage near zero under a reliable verifier and rising as verifier signal degrades, is specific to that deployment, we replicated the same ten experimental verifier and domain conditions through a configured Kimi K3 route. The returned provider model identity was not retained. We treat this as a qualitative sensitivity check, not as evidence that the reported magnitudes transfer across models. Every condition has 100% model-response completion: calls that returned a provider error were retried under the same model, prompt, temperature, caps and seed, never overwriting a completed call. We report all ten conditions rather than filtering by a reliability threshold. Repair, breakage, and net use all matched task$\times$seed pairs across the compared arms, with reservation rejections retained as workflow failures.

The qualitative ordering replicates across conditions: breakage is at or near zero under oracle tests and under tight self-check (0.000, 0.009, 0.019, 0.003), rises as visible tests are masked ($0.000 \to 0.060 \to 0.216$), peaks under the two no-signal conditions (0.146, 0.216). The signal-restored condition covers 68 of the 100 no-signal OOD tasks; restricted to that common subset both conditions use the same 189 reference-correct pairs, on which breakage is 0.132 without tests and 0.011 with them restored. These are separate runs on a matched task subset rather than a within-run restoration manipulation. The ordering is the same one GPT-4o shows in its signal ablations, though at different magnitudes (e.g. code/no-tests: 0.216 here against 0.235 there). We read this as a second-family replication of the qualitative verifier-sensitivity pattern, not of the controlled mechanism test. Neither check licenses magnitude transfer, which already fails within GPT-4o (§9).

P OFFICIAL GLM-4-FLASH PROVENANCE REPLICATION

We run the controlled provenance intervention through Zhipu's official API with model identifier `glm-4-flash-250414`. The server-returned model field was logged for completed responses and matched that identifier every time; the analysis below uses only the frozen result files. The provider lists this endpoint as free,[1] so both invoiced and ledger study cost are zero. Provider errors

[1] Official model page, accessed 2026-09-06.

Table 24: Kimi K3 replication of all ten experimental conditions. n_{eff} is the number of matched protected–baseline task×seed pairs; reservation rejections remain and are scored as failures. Repair, breakage, and net are point estimates with task-clustered percentile-bootstrap 95% CIs (the task is the resampling unit). Completion is the protected arm's fraction of task×seed calls that returned a model response.

Condition	p	n_{eff}	repair [95% CI]	breakage [95% CI]	net [95% CI]	completion
code, oracle, loose	0.771	450	0.204 [0.109, 0.315]	0.000 [0.000, 0.000]	+0.047 [+0.024, +0.073]	100%
code, oracle, tight	0.773	450	0.000 [0.000, 0.000]	0.009 [0.000, 0.020]	−0.007 [−0.016, +0.000]	100%
math, self-check, loose	0.807	450	0.092 [0.037, 0.161]	0.019 [0.006, 0.035]	+0.002 [−0.011, +0.016]	100%
math, self-check, tight	0.804	450	0.011 [0.000, 0.039]	0.003 [0.000, 0.009]	−0.000 [−0.007, +0.007]	100%
BBH, self-check, loose	0.950	360	0.000 [0.000, 0.000]	0.018 [0.006, 0.032]	−0.017 [−0.031, −0.006]	100%
code OOD, no tests	0.867	300	0.300 [0.143, 0.489]	0.146 [0.102, 0.195]	−0.087 [−0.133, −0.040]	100%
math OOD, self-check	0.440	300	0.060 [0.027, 0.098]	0.098 [0.046, 0.162]	−0.010 [−0.040, +0.020]	100%
code OOD, tests restored	0.926	204	0.467 [0.143, 0.882]	0.011 [0.000, 0.027]	+0.025 [−0.010, +0.064]	100%
code, 50% tests, loose	0.773	450	0.284 [0.179, 0.405]	0.060 [0.031, 0.095]	+0.018 [−0.020, +0.056]	100%
code, no tests, loose	0.773	450	0.186 [0.095, 0.292]	0.216 [0.167, 0.268]	−0.124 [−0.171, −0.080]	100%

are retained rather than scored as wrong. The math baseline completed all 150 tasks in each of three seeds, with accuracies 0.760, 0.773, and 0.780. The code baseline completed 147, 147, and 148 of 150 tasks; the other eight rows retain provider errors. Its accuracies among completed responses are 0.694, 0.680, and 0.682.

The primary mechanism populations are fixed by baseline outcomes before any arm outcome is examined. They contain 105 math tasks and 96 code tasks that the corresponding baseline solves in all three seeds, giving 315 and 288 matched task–seed pairs per arm. All 945 math rows and 864 code rows in these matched analyses were usable and had neither provider errors nor settlement overruns; reservation rejections remain scored as workflow outcomes. Intervals below average seeds within task and then resample tasks.

Table 25: GLM-4-Flash replication of the three-arm provenance test. Within each domain, all arms use the same stable baseline-correct tasks. Entries are task-clustered bootstrap 95% CIs.

Incumbent source	breakage	acceptance path	rejection/refinement path
Code: 96 tasks, 288 matched pairs per arm			
explicit reuse	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]
same-policy regeneration	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]	0.000 [0.000, 0.000]
different-policy regeneration	0.094 [0.049, 0.146]	0.059 [0.024, 0.101]	0.035 [0.007, 0.069]
Mathematics: 105 tasks, 315 matched pairs per arm			
explicit reuse	0.010 [0.000, 0.022]	0.000 [0.000, 0.000]	0.010 [0.000, 0.022]
same-policy regeneration	0.019 [0.006, 0.035]	0.000 [0.000, 0.000]	0.019 [0.006, 0.035]
different-policy regeneration	0.079 [0.044, 0.117]	0.003 [0.000, 0.010]	0.076 [0.041, 0.114]

Code. Explicit reuse and same-policy regeneration have zero observed breakage. Different-policy regeneration has breakage 0.094 [0.049, 0.146], consisting of 0.059 [0.024, 0.101] on the acceptance path and 0.035 [0.007, 0.069] after rejection and refinement. Its task-paired difference from either of the other arms is +0.094 [+0.049, +0.146].

Mathematics. Different-policy minus explicit-reuse breakage is +0.070 [+0.038, +0.105] and different-policy minus same-policy is +0.060 [+0.025, +0.098]. Same-policy minus explicit reuse is +0.010 [−0.006, +0.025] and is not distinguished from zero.

The replicated conclusion is narrow. Exact reuse closes acceptance-path regression in both domains and has lower observed breakage than a different first-draft policy. It does not make total breakage

impossible: the math reuse arm retains 0.010 breakage after false rejection and refinement, as Eq. 3 allows. The zero code breakage for same-policy regeneration is an empirical cell, not the structural guarantee provided by byte-identical reuse. The cross-domain differences also preclude an effect-size transfer claim.

The primary table conditions on stable correctness across three runs, whereas b in the main audit conditions on each realized baseline-correct execution. An aligned single-run sensitivity uses all 114 baseline-correct executions in the complete 150-task math run. Breakage is 0.018 for exact reuse, 0.044 for same-policy regeneration, and 0.123 for different-policy regeneration; acceptance-path breakage is 0, 0, and 0.009, respectively. This narrower single-run comparison preserves the ordering without treating stable-task conditioning as the main estimand.

Q SERVING-REGIME INTERVENTION

Same model weights (`Qwen/Qwen2.5-Coder-7B-Instruct`, revision `c03e6d3`), same physical GPU, same vLLM/torch/CUDA build, same 150 math tasks and seeds $0/1/2$, temperature 0 with the seed forwarded, and client response caching off throughout. Only the serving regime varies: serial (`max-num-seqs=` 1, one client worker), batched (`max-num-seqs=` 16, eight workers, with server metrics confirming seven to eight concurrent in-flight requests rather than only the setting), prefix caching (serial concurrency, cache on, hit rate rising from 0% to 39.5% in the server log), then a repeat of serial as a drift and reversibility check. Conditions ran one at a time, in that frozen order, with a single vLLM instance live.

Per-request logs did not separately serialize defaults inherited from the fixed vLLM build, so the identity claim is limited to the recorded payload fields under the stated server configuration. Identity is exact UTF-8 equality of the first generated draft; no normalisation. Outcome agreement is final workflow correctness, which is not the same quantity as first-draft correctness. Intervals resample the 150 task clusters.

Table 26: Serving-regime conditions against the serial reference. "payloads" counts pairs whose recorded first-draft model payload (messages, model, temperature, output limit, seed) is identical. Flips are final workflow outcomes, correct→wrong / wrong→correct.

condition	n	payloads	first-draft identity [95% CI]	outcome agree	flips
repeat serial (S2)	450	450	100.0% [100.0, 100.0]	100.0%	0 / 0
batched, all rows	450	450	50.0% [43.6, 56.4]	91.8%	19 / 18
batched, excl. recovered	444	444	50.0% [43.5, 56.6]	91.7%	19 / 18
batched, excl. non-overlapping	434	434	50.2% [43.6, 56.8]	91.5%	19 / 18
prefix cache	450	450	54.2% [46.0, 62.2]	93.1%	17 / 14

Six batched rows hit a one-time client-side tokenizer import race under eight worker threads and were rerun serially against the same batched server; a timing audit of the request windows finds sixteen batched rows in total, that six included, with no other request in flight while they ran. Excluding either set changes the identity rate from 50.0% to 50.0% and 50.2% and changes no flip count, so we report all rows in the main text. The experiment covers one model, one GPU and one inference stack; it manipulates the serving regime and attributes the difference to nothing finer, and it does not establish the source of the byte differences in the hosted GLM matrices.

R BUDGET-PARITY FOLLOW-UP

Both arms reuse the frozen mathematics reference cohorts of the operational matrix, unchanged, so the pairing base is identical. Both use identical per-call output limits and an identical eight-call verify–refine allowance; the regeneration arm additionally receives one drafting call and $1{,}536$ output tokens, settled before the suffix begins, so its suffix headroom equals assignment's. Every first rejection reached refinement in both arms and both cohorts. Because the suffix allowance is now the same in both cohorts, the two differ only in which frozen references they reuse, and we call them prior-`tight` and prior-`loose` rather than budget tiers.

Table 27: Assignment and same-policy regeneration on reference-correct positions. Same-policy rows are stratified by their own first-draft correctness. Post-rejection failure is the share of rejected positions that still end wrong.

cohort	first incumbent	n	accepted	acc. wrong	rejected	rej. wrong	post-rej. failure
prior-tight	assignment correct	347	308	0	39	5	12.8%
prior-tight	same-policy correct	322	290	0	32	8	25.0%
prior-tight	same-policy wrong	25	4	4	21	15	71.4%
prior-loose	assignment correct	343	305	0	38	7	18.4%
prior-loose	same-policy correct	325	293	0	32	3	9.4%
prior-loose	same-policy wrong	18	1	1	17	11	64.7%

Table 28: Measured cost, drafting separated from suffix. The total difference is not drafting alone: the arms are rejected at different rates, so their suffix paths differ as well.

cohort	arm	draft calls	draft out-tokens	suffix calls	suffix out-tokens
prior-tight	assignment	0	0	1,093	959,235
prior-tight	same-policy	450	316,886	1,123	993,045
prior-loose	assignment	0	0	1,113	977,127
prior-loose	same-policy	450	316,961	1,054	919,159

Preservation costs repairs, and the two cohorts disagree on how much. The paired repair-rate difference, assignment minus same-policy over reference-wrong pairs, is $-.039$ [-.143, +.067] in prior-tight and $-.093$ [-.179, -.017] in prior-loose: the first interval includes zero, the second excludes it. Assignment forgoes repairs it cannot attempt, since it issues no drafting call; this is the opportunity cost against the breakage reduction in Table 3, not a second estimate of it.

Assignment's advantage is concentrated on the positions where regeneration returned a wrong incumbent for a correct stored reference: 25 and 18 positions, on which same-policy ends correct 6 and 6 times while assignment, holding the stored reference at those same paired positions, ends correct 24 and 17 times. This is a paired comparison on identical positions, not a counterfactual for assignment receiving a wrong incumbent, which the design does not produce. Conditional on a correct first incumbent the two arms' post-rejection failure rates reverse direction across cohorts on 32–39 rejections each, so we make no claim about refinement recovery. All intervals in this appendix and in §7 are post-hoc descriptive and task-clustered; the study was frozen before execution but is not a preregistered confirmatory test.

Table 29: Complete budget-parity transitions, 450 pairs per cohort. A is assignment and S same-policy regeneration. Repair and break columns give numerators/eligible denominators; the corresponding conditional rate is their ratio. Δ is each arm minus its frozen reference, not A minus S. Cohort names identify prior reference sets; suffix allowances are equal.

cohort	repair A	repair S	break A	break S	Δ_A	Δ_S
math prior-tight	18/103	22/103	5/347	27/347	$+0.029$	-0.011
math prior-loose	18/107	28/107	7/343	15/343	$+0.024$	$+0.029$
code prior-tight	14/138	11/138	0/312	3/312	$+0.031$	$+0.018$
code prior-loose	12/136	18/136	0/314	2/314	$+0.027$	$+0.036$

Table 29 provides both sides of the transition identity for mathematics and code. These count summaries use final grading labels; paired repair-rate intervals are reported above and in the supplementary code-parity analysis. They do not convert rare-event nulls into equivalence.

S Observed terminal-path decomposition of the budget result

A refusal is an entry with reserve_rejected; it records an attempted node, not an executed node. We count only successful node executions when determining whether refinement preceded termination. Settlement overruns are a separate execution-failure category. A wrong fallback against

a correct reference need not have been damaged by refinement: the two initial answers can differ. This table describes observed paths, not the counterfactual outcome of an unfunded call.

Table 30: GPT-4o terminal paths, 450 pairs per cell. G_0/G_+ are breakages at refusal with zero/positive executed refinements; F denotes execution failure, C_+/C_0 completed paths with/without executed refinement.

		termination counts			breakage counts					
cell	pairs	refused	ref. $= 0$	failure	total	G_0	G_+	F	C_+	C_0
code/`tight`	450	102	96	0	72	45	2	0	0	25
code/`loose`	450	1	0	2	45	0	1	1	18	25

T PATH DECOMPOSITION OF THE VERIFIER-SIGNAL ABLATION

Breakage on reference-correct pairs, split by terminal path, at each masking level of the GPT-4o code/`loose` ablation. The three path counts sum to the total, which reproduces the aggregate rates quoted in §5. No execution in this ablation terminated before verification, so the K_{pre} term of Eq. 2 is empty here; it is not empty in the GLM math/`tight` cell.

Table 31: Terminal-path decomposition by verifier-signal level.

tests retained	ref.-correct	rejected	never verified	acceptance-path	rejection-path	total	rate
1.0	327	51	0	25/276	20/51	45	.138
0.5	327	50	0	26/277	21/50	47	.144
0.0	327	327	0	0/0	77/327	77	.235

U CODE-DOMAIN BUDGET-PARITY REPLICATION

The design of Appendix R applied to the two frozen code reference cohorts, whose six reference files are byte-identical to the code references of the operational matrix. All 1,800 rows completed and no suffix call was gated. Break and accept are counted over reference-correct pairs as elsewhere; the accompanying counterfactual analysis replays the operational caps and counts positions those caps would have refused, which is a ledger replay rather than an executed arm.

Table 32: Code-domain budget parity. Intervals are post-hoc descriptive and task-clustered; assignment records no breakage in either cohort, so those upper endpoints are degenerate and exclude nothing.

		asg./same-policy		asg. − same-policy, 95% CI	
cohort	n	break	accept	breakage	accuracy
code prior-`tight`	312	0/3	0/1	−.010 [-.026,+.000]	+.013 [-.007,+.033]
code prior-`loose`	314	0/2	0/0	−.006 [-.016,+.000]	−.009 [-.029,+.009]

Two readings must be kept apart. The outcome comparison is uninformative at this event rate: simulating the executed design against the effect size observed in mathematics gives power .19 in code/`tight` and .09 in code/`loose`, and .34 and .13 even against complete elimination, so neither cohort could have distinguished a vanished effect from a persisting one. The counterfactual analysis is informative: under the original caps code/`tight` same-policy would have been refused a planned call at 71 of 450 positions against assignment's 11, a reachability asymmetry that the initial-refinement column of Table 2 does not show because it arises in later iterations.

V SPECIFICATION OF THE PREREGISTERED PROTECTED ARM

This arm instantiates the verifier-gated acceptance of QualityFlow-style program synthesis (Hu et al., 2025b) inside our audit, so its row reports on paths what such designs report as a pass rate. The *protected* arm of Table 1 is distinct from the zero-call assignment intervention of §6: it generates its own incumbent rather than reusing a stored artifact. It drafts the strongest single-call baseline structure at temperature zero, returns that draft unchanged whenever the verifier accepts, and otherwise alternates temperature-0.7 refinement with re-verification. The iteration cap of three applies to the refine–verify back edge, so an execution in which every verification fails permits four refinements in addition to the initial drafting call; mathematics also uses model calls for verification. The reservation ledger can stop this graph earlier. Code verifies with the task's own visible tests and mathematics with the gold-free re-derivation check of §4. If the reservation ledger refuses a planned call, execution returns the solution currently in hand; it does not restore an earlier best answer, and no best-so-far retention is implemented in any arm — Appendix W replays one instead. The distinction from assignment matters for interpretation: protected's incumbent is a fresh temperature-zero draft, so $I = B$ does not hold for it and its acceptance-path breakage is not structurally zero — it is zero in the code cells as an executed outcome, not by contract.

W REVERT-ON-NON-ACCEPTANCE AGAINST ASSIGNMENT

We replay a fallback that returns the stored reference whenever the executed assignment workflow terminates without verifier acceptance. This is a QualityFlow-style protective acceptance rule (Hu et al., 2025b), not an oracle best-of-all-candidates selector. The replay uses the four *GLM operational assignment* cells in Tables 2 and 6; it does not use the self-drafting protected arm of Table 1. Every trace starts with an executed `assign` node. On these frozen graphs, a completed execution with a terminal verification is accepted; refused calls and loop exhaustion are not. The reference label determines the replay's correctness, while output SHA-256 digests determine byte changes. Neither requires new model calls.

Across 1,800 positions, the fallback changes 176 outputs byte for byte and 14 correctness labels. It avoids 8 breakages and forfeits 6 repairs. In math/`loose`, breakages fall from $8/343$ to $1/343$, but repairs fall from $14/107$ to $9/107$; the accuracy difference is $+.004$ [-.013, +.024]. No cell's accuracy interval excludes zero. Retention is therefore not redundant under assignment, and these operational replay results do not establish a uniform accuracy advantage.

Table 33: Revert-on-non-acceptance on GLM assignment records. "changed" reports byte/correctness changes separately. Repair and breakage denominators are reference-wrong and reference-correct pairs. Intervals are descriptive, task-clustered 95% bootstrap intervals for revert minus executed accuracy.

					repair		breakage		
cell	pairs	accept	no accept	changed	exec.	revert	exec.	revert	Δ accuracy
code/`tight`	450	381	69	43/1	17/138	16/138	0/312	0/312	$-.002$ [-.007, +.000]
code/`loose`	450	386	64	42/0	16/136	16/136	1/314	1/314	$+.000$ [+.000, +.000]
math/`tight`	450	330	120	6/1	0/103	0/103	1/347	0/347	$+.002$ [+.000, +.007]
math/`loose`	450	364	86	85/12	14/107	9/107	8/343	1/343	$+.004$ [-.013, +.024]

The math/`tight` cell retains its operational budget gates; the replay changes only the returned artifact, not which calls execute. It is not a budget-parity intervention. For self-drafting workflows, the same replay is not identified from logs that overwrite the initial draft: a separately stored reference is not necessarily that workflow's own incumbent.

The sample gross-value difference is $(\lambda\,\mathrm{Avoided} - \mathrm{Forgone})/450$ within each cell. For math/`loose` it is $(7\lambda - 5)/450$: the point estimate is positive above $5/7$, while the descriptive interval excludes zero only at $\lambda = 5$ among the displayed ratios. That result is conditional on this operational cell and chosen loss ratio. Summing all four cells gives $(8\lambda - 6)/1800$, a descriptive mixture of repeated tasks and regimes, not 1,800 independent trials or a common deployment population.

Table 34: Revert minus executed assignment: gross-value differences per pair at loss ratios λ. Values exclude inference cost (the replay changes no calls). Intervals use the same 10,000 task-clustered paired resamples as Table 33, and are descriptive, unadjusted across cells or ratios. λ^* is the sample break-even ratio, not a guaranteed utility threshold.

cell	avoided	forgone	λ^*	$\lambda = 1$	$\lambda = 3$	$\lambda = 5$
code/`tight`	0	1	–	-0.002 [-0.007, +0.000]	-0.002 [-0.007, +0.000]	-0.002 [-0.007, +0.000]
code/`loose`	0	0	tie	$+0.000$ [+0.000, +0.000]	$+0.000$ [+0.000, +0.000]	$+0.000$ [+0.000, +0.000]
math/`tight`	1	0	0/1	$+0.002$ [+0.000, +0.007]	$+0.007$ [+0.000, +0.020]	$+0.011$ [+0.000, +0.033]
math/`loose`	7	5	5/7	$+0.004$ [-0.013, +0.024]	$+0.036$ [-0.004, +0.084]	$+0.067$ [+0.002, +0.147]

X CLAIM–EVIDENCE MAP

Table 35 separates empirical evidence from structural relations and untested scope; counterexamples need not generalize across regimes.

Table 35: Claims, evidence, and status. No row is upgraded on the strength of a point estimate alone.

Claim	Evidence	Status
Net accuracy conceals opposing repair and breakage terms	GPT-4o frozen test: $-.033$ $[-.087, +.020]$ with repair .244, breakage .138 (Table 1)	Supported
Composite budgets change transitions and execution access	`loose`→`tight` moves repair .244 → .203 and breakage .138 → .220; 102/450 planned calls are refused, 96 before any refinement	Supported for the composite profile, not for a single resource
Verifier signal moves breakage	Masking visible tests $1.0 \to 0.5 \to 0.0$: breakage .138 → .235, net $-.118$ $[-.178, -.058]$; Kimi K3 reproduces the ordering	Supported
Changing the drafting policy inside a workflow regresses solved items	Different-policy breaks more in three operational cells, with unequal refinement access in two; 22 and 30 code breakages occur on acceptance paths (Table 2). Appendix P repeats the ordering	Conditional: operational effects, not equal-path causal contrasts
Stored assignment closes acceptance-path breakage	Structural consequence of $I = B$, verified as 0 acceptance-path breakages over four cells	Structural (verified contract)
Assignment reduces breakage relative to same-policy regeneration	Budget-parity follow-up, drafting funded outside the suffix allowance, no suffix call gated: $-.063$ $[-.099, -.031]$ (prior-`tight`); $-.023$ $[-.049, +.000]$ (prior-`loose`)	Conditional
Assignment improves net accuracy relative to same-policy regeneration	$+.040$ $[+.007, +.076]$ in prior-`tight` only; $-.004$ $[-.031, +.022]$ in prior-`loose`	Conditional, one cohort; no uniform advantage
Revert-on-non-acceptance trades repairs for preservation	On 1,800 GLM assignment positions it avoids 8 breaks and forfeits 6 repairs (Table 33); accuracy intervals include zero	Conditional: executed operational records
Assignment achieves at least as many repairs, or dominates in utility	prior-`loose` forgoes ten repairs, 18 against 28, and the repair-rate interval excludes zero	Counterexample to a uniform claim
Refinement recovers better under stored assignment	Conditional on a correct first incumbent, post-rejection failure is 12.8% against 25.0% and 18.4% against 9.4%, reversing across cohorts on 32–39 rejections	Unresolved
Serving configuration can change greedy outputs and final workflow outcomes	Same weights, GPU and recorded payload: batching reproduces 50.0% of first drafts and flips 8.2% of paired outcomes; prefix caching 54.2%	Conditional: one model, GPU and inference stack
Same-policy regeneration is equivalent to assignment	Refuted in prior-`tight` under budget parity, $5/347$ against $27/347$; the operational matrix's contrasts remain unresolved at its own power (§8)	Counterexample in prior-`tight`; no uniform relation
A byte-stable serving contract removes the distinction	Serial Qwen satisfies the tested byte-stability condition; batched and prefix-cached configurations do not. No general serving guarantee is claimed	Conditional
Magnitudes transfer across models or splits	Locked transfer to HumanEval+ and held-out math fails, MAE .061, sign agreement $6/8$	Not supported by the tested transfer
The frozen candidate can be certified at $\epsilon = .05$	Held-out bound $0.07699 > .05$, and the rule's floor at this sample size is 0.05065	Abstains: frozen rule cannot pass here
A shared false-rejection screen is a tighter certificate	Dominated by the candidate-specific bound at every candidate count we consider	Structurally no tighter; timing advantage only
Workflow search rediscovers the provenance choice	Preregistered search and two offline replays identify no mechanism (Appendix L)	Not recovered in the executed search
Preservation holds for repository or tool state	Not evaluated; our tasks are short-form program synthesis and mathematics	Untested / out of scope