REPAIR RATE IS NOT REPAIR: A COUNTERFACTUAL AUDIT OF CROSS-MODEL CODE CRITIQUE

Anonymous authors
Paper under double-blind review

ABSTRACT

Critique-and-revise pipelines are routinely credited with the improvements that follow them: a critic comments on a program, the program is revised, it passes its tests, and the critic is scored as having repaired it. That is a causal claim, almost never tested against the obvious control: what the same reviser would have produced from the same program given no substantive advice. We run that control. In an existing three-round experiment on HumanEval+ whose round-0 programs are shared byte-identically across conditions, we form matched pairs at the only transition where the starting state is fixed and compare each advice condition against a null-advice arm whose critique is the fixed string `NO ISSUES FOUND`. For a neutral cross-model critic, the naive repair rate reports 0.034 while the counterfactual uplift is -0.041 (95% CI $[-0.061, -0.022]$); with an added "senior engineer" persona the figures are 0.011 and -0.037 (95% CI $[-0.055, -0.018]$). The naive statistic is computed only over initially failing programs. On that stratum the causal uplift is $+0.011$ for the neutral critic and -0.011 for the persona variant, neither resolvable away from zero, so neither is what drives the inversion. What the naive rate excludes by construction is the already-passing stratum, where essentially all of the harm falls. A pipeline scored by repair rate therefore records a benefit while the full matched comparison records a harm. The mechanism is visible in the data: the cross-model critic reports a defect in 94.5% of code that already passes its tests, and breaks such programs at 5.2% against a 0.2% null-advice baseline. Harmful-to-helpful flip ratios are $21{:}1$ for the neutral critic and $10{:}1$ for the persona variant, $41{:}3$ pooled. These are measurements of one model pair on one benchmark, and our self-critique arm is degenerate and supports no conclusion about self-critique.

1 INTRODUCTION

A large family of language-model pipelines has the same shape. A model produces a candidate artifact, a critic—the same model, a different model, or a tool—comments on it, and the model revises in light of the comment. When the revision is better than the original, the critique is credited with the improvement (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024; Xu et al., 2024a). In code generation, where correctness is executable and therefore cheap to measure, this crediting is usually made numerical: one reports the fraction of initially failing programs that pass after critique-and-revision, and calls it a repair rate.

The repair rate is a causal claim wearing the clothes of a descriptive statistic. To say that a critique repaired a program is to say that the program would not have been repaired without it. That comparison requires a counterfactual arm—same reviser, same starting program, no substantive advice—and the repair rate does not contain one: it conditions on the advice having been given, so any repair the reviser would have performed anyway, simply by being asked to look again, is silently attributed to the critic. So is the possibility running in the other direction: a critique that breaks a program that was already correct never appears in a repair rate at all, because a repair rate is computed only over initial failures.

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.

This paper runs the missing control. We re-analyze an existing three-round critique-and-revise experiment on HumanEval+ (Chen et al., 2021; Liu et al., 2023) in which, by construction, round-0 programs are generated once per seed and handed byte-identically to every condition. That structural fact makes the round-0 to round-1 transition a matched counterfactual: identical starting code, different advice, same reviser, same decoding parameters. Against each advice condition we place a *null-advice* arm whose revision prompt receives the fixed string `NO ISSUES FOUND` with no critic invoked. It measures what revision alone accomplishes; the difference between the two measures what the advice accomplishes.

Under that comparison every matched pair falls into one of four cases, according to whether the outcome differed and in which direction: the advice fixed something the null arm would have left broken, the advice was present but the null arm would have passed anyway, the advice broke something the null arm would have kept working, or neither arm passed. Only the first is repair in the sense the word is usually meant.

Three numbers set up the result. First, the cross-model critic is not selective: on code that already passes its unit tests it declines to report a defect only 5.5% of the time, a false-alarm rate of 0.945, rising to 0.989 once a "senior engineer" persona is added to the critic prompt. Second, the resulting edits break far more than they fix: 21 harmful flips to 1 genuine fix for the neutral critic and 20 to 2 with the persona, $41{:}3$ pooled. Third, and this is the finding the title refers to, the naive repair rate for the neutral cross-model critic is $+0.034$ while its counterfactual uplift is -0.041 (95% CI $[-0.061, -0.022]$). The two differ in sign, and the resolvable part of that difference is not about the initially failing programs: restricted to those same 87 instances, the causal uplift is $+0.011$ for this critic and -0.011 for the persona variant, neither resolvable away from zero (Section 5.1). What drives the overall inversion instead is what the naive rate never examines at all, since it is computed over initial failures alone while the harm falls almost entirely on programs that were already correct. A pipeline scored by repair rate therefore records this critic as mildly helpful; measured against its own null-advice control over the whole population, it is reliably harmful.

The harm is also not distributed uniformly. Almost all of it falls on programs that were already correct—the cross-model critic breaks 5.2% of passing programs against a null-advice baseline of 0.2%—while failing programs are close to inert under any advice condition, consistent with a critic that cannot tell correct code from incorrect and whose comments act as undirected perturbation.

This is a measurement paper: we propose no method, and the contribution is a decomposition, the numbers that come out of it, and an explicit account of what the data cannot support. In particular, our self-critique condition is degenerate—the self-critic emitted the null string on every one of its $1{,}476$ opportunities—so we draw no conclusion about self-critique, and we flag this wherever that condition appears. Our claims are scoped to one model pair, one benchmark, one temperature, and the single transition at which the counterfactual is valid.

2 Related work

Self-refinement and iterative critique. Self-Refine (Madaan et al., 2023) has a model provide feedback on its own output and revise accordingly; Reflexion (Shinn et al., 2023) keeps verbal self-reflections in an episodic buffer; CRITIC (Gou et al., 2024) grounds the critique in external tools; LLMRefine (Xu et al., 2024a) replaces free-form critique with a learned error locator; and in code, Self-Debug (Chen et al., 2023) teaches a model to explain and repair its own programs. These works differ in where the feedback comes from but share an evaluation convention: the pipeline with critique is compared against one with no revision step, bundling revision and advice and crediting them jointly. Our null-advice arm separates them.

Skeptical results. A parallel line reports that the improvements are fragile or absent. Huang et al. (2024) find that models cannot reliably self-correct reasoning without external grounding, performance sometimes degrading after self-correction. Stechly et al. (2024) observe collapse under self-critique while sound external verification helps, locating the problem in the verifier, not the corrector. Tyen et al. (2024) make the same split explicit: models are poor at finding errors but competent at fixing them once told where the error is. Xu et al. (2024b) show that the self-refinement loop amplifies a model's bias toward its own generations. Closest to our setting, Olausson et al. (2024)

ask whether self-repair is a silver bullet for code generation and find gains that are modest, uneven across data subsets, and sometimes absent once the cost of repair is counted.

Our contribution relative to this line is the control arm, not the skepticism: these works compare against no-revision baselines or sampling budgets, while we hold the starting program byte-identical and vary only the advice, separating the two failure modes a pooled comparison confounds — advice that is inert, and advice that is actively destructive — visible only under that separation (Section 5.1).

Counterfactual attribution for model pipelines. The counterfactual framing itself is not new, and we do not claim it. CausalFlow (Bonagiri et al., 2026) is the closest prior work: it performs step-level interventional attribution on failed agent traces, computing causal responsibility scores by intervening on individual steps, and then generates minimally edited corrections that double as training signal. The overlap is real: both replace correlational credit assignment with an interventional comparison over matched states, but differ in the object of attribution. CausalFlow asks which step inside a trace caused a failure that already occurred; we ask the prior question of whether the critique channel carries positive causal value at all, against a null-advice control holding reviser and starting state fixed. It assumes the corrective signal is worth attributing and localizes it; we test whether there is signal to localize, with a natural composition applying step-level attribution only to advice already shown non-negative causal value.

More broadly, the design is an ordinary matched-pair potential-outcomes comparison in the sense of Rubin (1974), formalized in Section 4; our uncertainty estimates use the nonparametric bootstrap (Efron, 1979), clustered by problem.

3 Experimental setup

Benchmark. All experiments use HumanEval+ (Liu et al., 2023), the test-augmented version of HumanEval (Chen et al., 2021), comprising 164 Python problems. Correctness is executable: a program passes if and only if it survives the problem's unit tests, so the outcome needs no model-based judgment. Every number in this paper derives from that binary pass signal.

Models and decoding. The generator and reviser (model A) is Qwen2.5-Coder-7B-Instruct (Hui et al., 2024). The cross-model critic (model B) is deepseek-coder-6.7b-instruct (Guo et al., 2024). Both are loaded 4-bit quantized and decoded at temperature 0.2. The critic is ungrounded: it sees only the problem statement and the candidate source text, with no execution trace, test output, or tool access. The pipeline runs three critique-and-revise rounds under three seeds, 1234, 5678, and 9012, giving $164 \times 3 = 492$ problem instances per condition per round.

Conditions. Five conditions share the same round-0 program and differ only in what the reviser is told.

- **C0**, no critique. Frozen at round 0 and never revised; it is the do-nothing reference.
- **C4**, null critique. The revision prompt receives the fixed string `NO ISSUES FOUND` and no critic model is called. The reviser still runs. This is the counterfactual arm.
- **C1**, self-critique. Model A critiques its own code.
- **C2**, cross-model critique. Model B critiques, under a neutral prompt.
- **C3**, cross-model critique with persona. Model B critiques under an injected "senior engineer" persona.

C4 is the arm that makes the rest interpretable: it removes only the informational content of the critique, so whatever C4 achieves is achievable without a critic. C0 additionally removes the revision step, separating the cost of revising at all from the cost of revising on advice.

The matched-pair structure. Round 0 is generated once per seed and the same record starts all five conditions. We verified this rather than assuming it: the round-0 program is byte-identical across all five conditions for $164/164$ problems in all three seeds, and the pipeline asserts the property and aborts if it fails, because the entire comparison rests on it.

The transition from round 0 to round 1 is therefore a genuine matched counterfactual: both arms begin from the same characters of source code and are revised by the same model at the same temperature, the only difference being the advice string. From round 1 onward each condition continues from its own previous output, so by round 2 the arms occupy different states and any comparison between them confounds the advice with the divergence. We therefore restrict every causal quantity to round $0 \rightarrow$ round 1, retaining rounds 2 and 3 for descriptive use only and marking them as such wherever they appear.

Merged dataset. The merged trajectory table has $9{,}840$ rows, one per (seed, condition, round, problem): $3 \times 5 \times 4 \times 164$. Round-0 pass@1 is $135/164 = 0.823$ in each seed; of the 492 pooled round-0 instances 405 pass and 87 fail, the imbalance behind the power asymmetry of Section 6.2. Because C0 is frozen, its rows at rounds 1–3 are reconstructed by copying the round-0 record forward with the regression, repair, and churn flags cleared, matching the original pipeline.

4 THE A/B/C/D COUNTERFACTUAL DECOMPOSITION

4.1 DEFINITIONS

Fix a seed s, a problem t, and an advice condition $X \in \{\text{C1}, \text{C2}, \text{C3}\}$. Let $Y^X_{s,t} \in \{0, 1\}$ be the round-1 pass indicator under X and $Y^{\text{C4}}_{s,t}$ the round-1 pass indicator under null advice, both starting from the same round-0 program. Because the starting state is shared, the pair $(Y^{\text{C4}}_{s,t}, Y^X_{s,t})$ is a realized pair of potential outcomes in the sense of Rubin (1974), and we can classify it directly:

	Bucket	Condition	Interpretation
A	`true_helpful`	$Y^{\text{C4}} = 0,\ Y^X = 1$	the advice caused the fix
B	`false_helpful`	$Y^{\text{C4}} = 1,\ Y^X = 1$	would have passed anyway
C	`harmful`	$Y^{\text{C4}} = 1,\ Y^X = 0$	the advice caused the break
D	`true_useless`	$Y^{\text{C4}} = 0,\ Y^X = 0$	nothing that mattered changed

Only bucket A is repair in the sense the word is normally used, and bucket B is the category a naive repair rate cannot distinguish from A. Bucket C is not straightforwardly invisible to the rate: when it originates from an already-passing program the rate never evaluates that instance, since it conditions on initial failure; when it originates from an initially failing program the rate evaluates the instance but can only record it as an unrepaired failure, unable to isolate the advice's contribution without a null-arm comparison.

We compare two estimands: the **naive repair rate**, the quantity the literature usually reports, and the **causal uplift**, the net effect of the advice against its own null arm,

$$\widehat{r}_{\text{naive}}(X) = \Pr\big[\, Y^X = 1 \mid \text{round-0 failed} \,\big], \qquad \widehat{u}(X) = \Pr[A] - \Pr[C]. \tag{1}$$

Equation 1 has a convenient algebraic identity. Writing $\mathbf{1}[\cdot]$ for the bucket indicators, $\mathbf{1}[A] - \mathbf{1}[C]$ is $+1$ exactly on A, -1 exactly on C and 0 on B and D, so per pair it equals $Y^X - Y^{\text{C4}}$: the causal uplift is exactly the mean paired difference in pass rate between the advice arm and the null arm. The bucket decomposition and the paired difference are two views of the same estimand, the buckets carrying the extra information of which direction each flip went.

The two estimands have different denominators, and this matters for how they may be compared. The naive repair rate conditions on initial failure, so it is computed over the 87 failing instances only; the causal uplift is computed over all 492 pairs, including the 405 that started from a passing program. Section 5.1 reports the uplift restricted to the failing stratum as well, so that the two can also be read on a common population.

Uncertainty. Confidence intervals come from a paired cluster bootstrap (Efron, 1979) with $10{,}000$ resamples at a fixed random seed of $20{,}250{,}910$. Because each problem appears once per seed, the resampling unit is the problem: we average the paired difference within each problem across seeds, then resample problems with replacement, since treating the 492 pairs as independent would understate the interval width. Point estimates are reported at the pair level, and for the pooled subset they coincide exactly with the bootstrap center.

Table 1: Pooled A/B/C/D decomposition of the round-0 → round-1 transition. $n = 492$ matched pairs per condition (3 seeds × 164 problems). Proportions are pooled over seeds; σ is the sample standard deviation (ddof=1) of the proportion across the three seeds. C1 is degenerate (Section 6.1) and is shown for completeness only.

	A `true_helpful`		**B** `false_helpful`		**C** `harmful`		**D** `true_useless`	
Condition	n	$p\ (\sigma)$	n	$p\ (\sigma)$	n	$p\ (\sigma)$	n	$p\ (\sigma)$
C1 (self, degen.)	1	0.002 (0.004)	406	0.825 (0.004)	0	0.000 (0.000)	85	0.173 (0.004)
C2 (cross-model)	1	0.002 (0.004)	385	0.783 (0.009)	21	0.043 (0.006)	85	0.173 (0.007)
C3 (cross + persona)	2	0.004 (0.004)	386	0.785 (0.009)	20	0.041 (0.013)	84	0.171 (0.000)

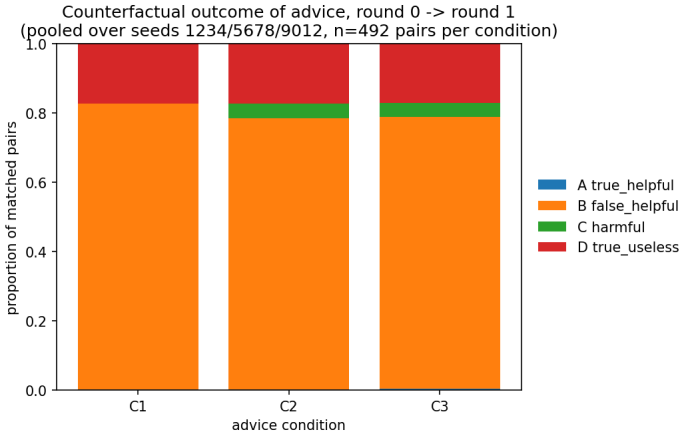


Figure 1: Counterfactual outcome of advice at the round-0 → round-1 transition, pooled over the three seeds. Bucket A (true repair) is too small to be visible at this scale in every condition, which is the point.

4.2 Bucket counts

Table 1 gives the pooled decomposition; Figure 1 shows the same proportions. The dominant observation is that most pairs land in B or D, where the advice did not change the outcome: $491/492$ (99.8%) for C1 and $470/492$ (95.5%) for each cross-model condition. What remains is lopsided: the neutral cross-model critic produces 21 harmful flips against 1 genuine fix and the persona variant 20 against 2, ratios of 21:1 and 10:1, or 41:3 pooled. Across-seed variation is small, the C2 harmful count being 6, 7 and 8.

The C1 row is not a finding about self-critique: as Section 6.1 documents, that critic emitted the null string on every instance, making C1 an unintentional replicate of C4, and its single A-bucket pair is one problem flipping under stochastic decoding.

5 Results

5.1 The naive repair rate inverts the sign of the effect

Table 2 is the main result, placing the naive repair rate beside the causal uplift against the null-advice arm; Figure 2 plots the same quantities.

For C2 the naive repair rate is 0.034 and the causal uplift -0.041, 95% interval $[-0.061, -0.022]$: a gap of 0.075, but the consequential fact is the sign, since a pipeline scored by repair rate would record C2 as mildly beneficial while the matched comparison shows it reliably harmful. C3 behaves the same way, 0.011 against -0.037, interval $[-0.055, -0.018]$, a gap of 0.048.

Table 2: Naive repair rate versus causal uplift, round-0 → round-1, pooled over three seeds. Naive repair rate is $\Pr[Y^X = 1 \mid \text{round-0 failed}]$, over 87 failing instances. Causal uplift is $\Pr[A] - \Pr[C]$ over all 492 pairs. Intervals are paired cluster bootstrap 95% CIs, 10,000 resamples, clustered by problem. *Gap* is the arithmetic difference between the two columns, which are scoped to different populations (87 failing instances versus all 492 pairs); it measures how far apart the two reported statistics are, not an attribution error within either.

Condition	Naive repair	95% CI	Causal uplift	95% CI	Gap
C1 (self, degen.)	0.023 (2/87)	$[0.000, 0.061]$	$+0.002$	$[0.000, +0.006]$	0.021
C2 (cross-model)	0.034 (3/87)	$[0.000, 0.081]$	$\mathbf{-0.041}$	$[-0.061, -0.022]$	0.075
C3 (cross + persona)	0.011 (1/87)	$[0.000, 0.030]$	$\mathbf{-0.037}$	$[-0.055, -0.018]$	0.048

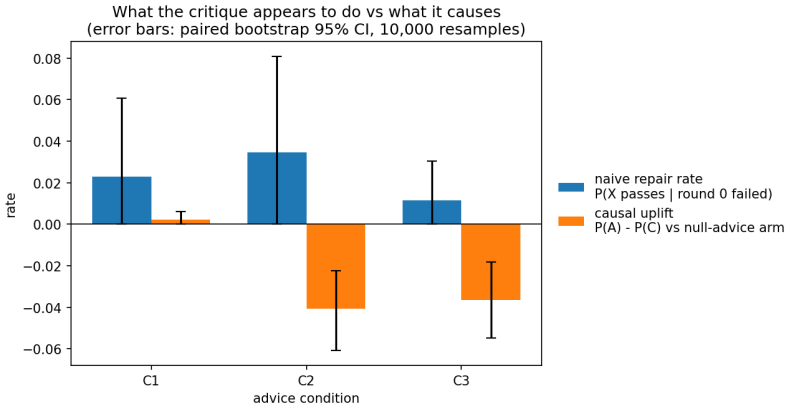


Figure 2: Naive repair rate against causal uplift per condition; error bars are paired cluster bootstrap 95% intervals over 10,000 resamples. The bars are scoped differently: the naive bar covers the 87 failing instances, the causal bar all 492 pairs. See Section 5.1 for the stratum-restricted comparison.

These two statistics are computed over different populations. Restricted to the failing stratum — the same 87 instances the naive rate uses — C2's naive rate is 0.034 against a causal uplift of $+0.011$, interval $[0.000, +0.030]$, a positive point estimate whose lower bound sits exactly at zero and is therefore not resolvable away from it; the $2/87$ gap is bucket B, repairs the null arm would have made anyway that the naive rate credits to the critique, so even here the naive figure overstates the critique's contribution, though the point estimates agree in sign. C3's naive rate is 0.011 against a causal uplift of -0.011, interval $[-0.061, +0.030]$, which straddles zero: here the point estimates disagree in sign, since the naive rate can only register repairs and has no way to register C3's two harmful flips as harm. Neither failing-stratum effect drives the paper's headline result: the statistically resolvable overall sign inversion is driven by the already-passing stratum, which the naive rate excludes by construction and in which almost all of the harm occurs — the only exceptions are C3's two harmful flips that started from failing programs (Section 5.2).

The null arm supplies the reference floor. Revision under null advice repairs $2/87 = 0.023$ of round-0 failures on its own and breaks $1/405 = 0.002$ of round-0 passes, so C2's naive 0.034 is barely distinguishable from what the reviser does with no critic at all and C3's 0.011 is below it: the apparent repairs are mostly revision, not critique.

Note also the asymmetry in what the intervals resolve. Both cross-model causal intervals exclude zero on the harmful side; all three naive-repair intervals have their lower bound exactly at zero and so are not resolvable away from it. The only statistically resolvable effect here is that cross-model critique hurts, for the structural reason given in Section 6.2.

Table 3: Causal uplift split by whether the round-0 program already passed its tests. Bootstrap intervals as in Table 2. The two regimes behave very differently: the harm is almost entirely concentrated in the already-passing stratum, the only exceptions being C3's two harmful flips from failing programs.

Condition	**Round-0 state**	n	$\Pr[A]$	$\Pr[C]$	**Causal uplift**	**95% CI**
C1 (degenerate)	failed	87	0.000	0.000	0.000	$[0.000, 0.000]$
C1 (degenerate)	passed	405	0.002	0.000	$+0.002$	$[0.000, +0.007]$
C2	failed	87	0.011	0.000	$+0.011$	$[0.000, +0.030]$
C2	passed	405	0.000	0.052	$\mathbf{-0.052}$	$[-0.075, -0.029]$
C3	failed	87	0.011	0.023	-0.011	$[-0.061, +0.030]$
C3	passed	405	0.002	0.044	$\mathbf{-0.042}$	$[-0.064, -0.024]$

Table 4: Mechanism of the 41 harmful flips, assigned by reading each case (21 for C2, 20 for C3). Counts only; the sample is far too small for inference.

Mechanism	**C2**	**C3**	**All**	**Mechanism**	**C2**	**C3**	**All**
Hallucinated bug	10	7	17	Critique text as code	1	1	2
Spec misunderstanding	5	3	8	Anchoring, no edit	0	2	2
Defensive logic	3	3	6	Semantic drift	1	0	1
Unclear	1	2	3	Valid issue, wrong fix	0	1	1
				Broad rewrite regression	0	1	1

5.2 Almost all of the damage lands on code that was already correct

Splitting the decomposition by the round-0 outcome is the sharpest result in the study (Table 3; the corresponding visualization is Appendix F).

On already-passing code the cross-model critic breaks 5.2% (C2) and 4.4% (C3) of programs against a null-advice breakage of 0.2%, both intervals excluding zero. On already-failing code the advice is close to inert: 84 to 85 of the 87 failing pairs per condition remain in bucket D regardless of what the critic said, and neither cross-model interval in that stratum is resolvable away from zero: C2's lower bound sits exactly at zero and C3's straddles it.

Every one of C2's 21 harmful flips started from a round-0 program that passed, and for C3, 18 of 20 did, the remaining two from an assertion failure: the critic is not converting one kind of failure into another, and in those two it suppressed a revision the null arm used to repair the program.

The critic is not selective. The mechanism is visible in the critic's own output. On code that already passes its unit tests the neutral cross-model critic declines to report a defect in only 5.5% of cases, a false-alarm rate of 0.945; the "senior engineer" persona raises this to 0.989, making the critic more confident and less discriminating at once. On the 87 programs that actually fail, the same critics report a defect at 0.989 (C2) and 1.000 (C3) — essentially the same rate at which they flag already-correct code; all four rates are pooled over the three seeds.

A critic whose defect-reporting rate barely moves between correct and broken code carries almost no information about which programs are actually wrong, and the reviser has no way to discount it. This is the separation Tyen et al. (2024) and Stechly et al. (2024) identify in reasoning tasks—error location is the hard part, not error correction—appearing here as a measurable cost.

Descriptive round-3 pass@1 figures, which move in the same direction, are reported in Appendix G.

5.3 How the harmful edits happen

We read all 41 harmful flips individually — round-0 program, critique, revision, diff and failure output — and labelled each (Table 4; definitions in Appendix E).

Three cases, one per leading mechanism. `HumanEval/48` (C2, hallucinated bug): the critic called `is_palindrome('a')` wrong, though one character is a palindrome; the reviser added `text != '' and ...`, failing the empty string the tests expect to pass. `HumanEval/80` (C3, spec misunderstanding): the critic wanted `is_happy('a')` to be `True`, but the specification requires length ≥ 3; the reviser flipped `return False` to `return True`. `HumanEval/30` (C3, defensive logic): the critic speculated about strings in the list and the reviser added `isinstance(x, int)`, dropping valid numeric values; the same edit appears in two other runs.

In 39 of 41 cases the critic did not mark the code clean, so these edits follow a reported defect, and asserting a defect that is not there or misreading the problem statement account for 25 of 41. The edits are small (median diff 2 lines against 0 for C2/C3 round-1 revisions that did not flip): in this sample the harm comes from short, confident, wrong corrections rather than sprawling rewrites. Two caveats. The two *critique text as code* cases are not critique harm: the critic's `NO ISSUES FOUND` arrived inside a code fence and was pasted in as the function body, a failure of output handling rather than of advice (Section 6.3), and they are the only two the critic marked clean. The two *anchoring, no edit* cases are the same task under two seeds, so they are one problem, not two independent observations.

Critique length carries no signal. Longer critiques were not more useful: median critique length in the harmful bucket is 734 characters against 1,042 in the useless bucket for C2, and 802.5 against 1,106.5 for C3, with harmful critiques if anything shorter. The true-helpful bucket holds 1 pair for C2 and 2 for C3, too few to interpret; nothing here supports critique length as a stand-in for critique value (Appendix C).

6 Limitations

6.1 The self-critique arm is degenerate and supports no conclusion

The most important limitation of this study concerns condition C1, and it is a defect in the data rather than in the analysis. **C1 is not a self-critique condition but an unintentional second copy of the null-advice arm.**

Across all three seeds and all three rounds, every one of the 1,476 critiques produced by the self-critic is literally the null string: either `NO ISSUES FOUND` (15 characters) or `NO ISSUES FOUND.` (16 characters, the model appending a period). Asked to critique its own code, Qwen2.5-Coder-7B-Instruct never once reported a problem, at any round, under any seed. This is why the C1 critique lengths in Appendix C are 15 to 16 characters, and why C1's false-alarm rate is exactly 0.000: that rate is not evidence of a well-calibrated self-critic but an artifact of a critic that never says anything.

The consequence is that C1 and C4 receive byte-identical revision prompts up to a trailing period, and they behave accordingly. At round 1 the C1 and C4 programs are byte-identical on 163/164, 162/164, and 163/164 problems in seeds 1234, 5678, and 9012, with identical pass/fail outcomes on 164/164, 164/164, and 163/164. The +0.002 causal uplift reported for C1 in Table 2 is a single problem flipping in one seed, which is decoding noise, not evidence about self-critique.

We therefore make no claim about self-critique anywhere in this paper. The tempting reading of Table 2—that self-critique is harmless while cross-model critique is harmful—is not supported, because the condition never produced a critique to evaluate. All the data supports is that *this* self-critic, under *this* prompt, never engaged, so C1 measures the null condition twice. Separating "self-critique is safe" from "the prompt failed to elicit a critique" requires a C1 arm that actually emits critiques, and that experiment has not been run. We retain C1 in the tables because removing it would hide a real property of the run, and we label it as degenerate everywhere it appears.

6.2 Statistical power is asymmetric between the two directions

The design is well powered to detect harm and poorly powered to detect benefit: 405 of the 492 pooled round-0 instances already pass, leaving only 87 per condition that carry information about repair, backed by 33 distinct problems. Every naive-repair interval in Table 2 has its lower bound exactly at zero as a result, and so is not resolvable away from it. We therefore never describe a null

finding here as showing "no benefit": the experiment does not resolve whether cross-model critique repairs failing programs at a low rate, while it does resolve that it breaks passing programs well above the null-advice floor. Measuring the helpful direction with comparable precision would need a substantially weaker generator.

6.3 Scope of the configuration and of the causal claim

Every number describes the single configuration of Section 3, and we claim no generality beyond it. The critic is smaller than many deployed critics and comparable in capability to the model it critiques, and Olausson et al. (2024) report that a substantially stronger critic behaves differently; whether the sign inversion survives one is an open question this data cannot answer. The persona was a single prompt variant, and we can say it raised the false-alarm rate from 0.945 to 0.989 without saying which feature did so. Quantization may also affect the critic's calibration specifically rather than uniformly, and we ran no unquantized control. The critic here is also ungrounded (Section 3); the harm rate on already-correct code may be specific to this text-only setting and not to execution-grounded debugging (Zhong et al., 2024) or search-based self-correction (Zhou et al., 2024).

The mechanism labels are a single-annotator categorization. The labels in Section 5.3 were derived, with AI assistance, from case-level inspection of all 41 harmful flips using the stored critique text, the round-0 and round-1 programs, the diffs and the failure output, against a category list fixed in advance and extended twice where the evidence required it. Because the categorization was not independently annotated by multiple raters, the frequencies in Table 4 should be read as descriptive of this sample rather than as a validated prevalence estimate: there is no adjudication protocol and no measure of how stable the categories would be under a different reader. Three cases are `unclear` at low confidence because the critique could not be connected to the edit.

The null arm and the advice arms differ in output handling. The null arm's `NO ISSUES FOUND` is injected as plain text, whereas critic output reaches the reviser as generated; in two harmful flips an identically worded verdict arrived inside a Python code fence and was pasted into the program as its body. The arms therefore differ in parsing path as well as advice content, which the design does not control for. The evidence bounds this at 2 of 41; we leave the estimand alone, since dropping cases whose mechanism is inconvenient is the selection this paper argues against.

A single transition. Only the round-0 $\rightarrow$ round-1 transition is matched, so rounds 2 and 3 are reported descriptively and labelled as such, and the paper measures the causal effect of *one* round of critique. Whether the harm compounds, saturates, or partially self-corrects over further rounds a single matched transition cannot establish, though the descriptive round-3 figures are not in tension with compounding harm.

The outcome is also binary test-passing: a critique that improved readability, removed a latent bug HumanEval+ does not exercise, or made the code more maintainable would score as bucket D, or bucket C if it perturbed behavior. Our measurements concern functional correctness under the benchmark's tests and nothing else.

7 Conclusion

Crediting a critic with the improvements that follow its critique requires a counterfactual the standard repair rate does not contain. Supplying one changes the accounting qualitatively: the naive rate reports a benefit where the matched comparison finds a harm, not because it misjudges the failing programs it examines, where for this critic the two agree in sign, but because it never examines the passing ones.

Two practices follow. First, report a null-advice arm, not merely a no-revision baseline; it invokes no critic, and without it the effect can carry the wrong sign. Second, report outcomes on inputs that were already correct: a repair rate conditions on initial failure and cannot go down. The magnitudes will not transfer, but the methodological point should: repair rate is not repair, and the gap can reverse the conclusion.

REPRODUCIBILITY STATEMENT

The analysis in this paper is a deterministic re-analysis of stored checkpoints from an already-completed generation run; no model inference was performed for this paper. Section 3 specifies the benchmark, both models, the quantization setting, the decoding temperature, the three seeds, and the five conditions. Section 4 gives the bucket definitions, both estimands, and the bootstrap procedure, including the resampling unit and the fixed random seed ($20{,}250{,}910$, $10{,}000$ resamples). The matched-pair assumption on which every causal quantity depends is stated in Section 3 and is asserted programmatically at load time, with the pipeline aborting if round-0 programs are not byte-identical across conditions; it holds at $164/164$ in all three seeds. Appendix B lists every analysis decision made where the stored data was ambiguous, including the error-type parser, the treatment of the frozen C0 arm, and the choice of cluster bootstrap. Appendix D describes the derived tables from which every number in every table in this paper is taken.

ETHICS STATEMENT

This work involves no human subjects, no personally identifying data, and no new data collection. It re-analyzes model outputs on HumanEval+, a public benchmark of synthetic programming problems, using two publicly released open-weight code models. Executed code was confined to the benchmark's own unit tests.

The primary ethical consideration we see is the risk of over-generalization in the other direction. Our results are negative for one critic on one benchmark, and we have tried throughout to scope them accordingly rather than license a general claim that LLM critique is harmful. We also note explicitly, in Section 6.1, that one of our five conditions is degenerate, so that no reader draws a conclusion about self-critique from a condition that never produced a critique. We see no foreseeable harmful application of the measurement methodology itself, which is a control arm.

AI USE STATEMENT

We used generative AI tools for several parts of this work and describe each below. The research ideation was not AI-assisted: the research question, the counterfactual framing, and the A/B/C/D decomposition originated with the author. The experimental design was likewise not AI-assisted: the choice of generator, critic, benchmark, conditions, seeds, and decoding parameters were made by the author, and the author executed all generation runs personally. We did use an AI coding assistant to write the generation, critique, and revision pipeline notebooks and the analysis scripts, and to prepare the LaTeX source, including the tables and figure inclusion. We used AI assistance for literature search, to identify and retrieve candidate related work; every citation was then verified by the author against arXiv or the ACL Anthology before inclusion, and no citation was included on the strength of an AI system's recall alone. We also used an AI assistant for the mechanism categorization reported in Section 5.3: the 41 harmful cases were read and labelled with AI assistance against a category list we fixed in advance, and the resulting labels are disclosed in Section 6.3 as a single-annotator categorization rather than a validated taxonomy. We also used an AI assistant to draft the prose of this paper — the abstract, introduction, related work, experimental setup, results, limitations, and conclusion — working from the author's analysis outputs and the author's specifications, after which the author reviewed and revised the text across several rounds. This was drafting, not copy-editing, and we state it plainly.

Verification was also AI-assisted, and we describe it as such. The verification script was written with AI assistance and then reviewed and re-run by the author, and the review passes over the manuscript were AI-assisted with the author adjudicating each change. Every numeric claim in the paper is checked against its source CSV by that re-runnable script, which currently performs 232 checks covering table cells, prose-only derived quantities, and the interval and quantifier language used to describe them; the matched-pair assumption on which the entire analysis rests is asserted programmatically at data-load time and aborts the pipeline if violated. We have reviewed all AI-assisted work, and we take responsibility for the final content of this paper, including all text, claims, and artifacts.

REFERENCES

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

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.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, et al. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations (ICLR)*, 2024.

Donald B. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688–701, 1974.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024.

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13894–13908, 2024.

Wenda Xu, Daniel Deutsch, Mara Finkelstein, Juraj Juraska, Biao Zhang, Zhongtao Liu, William Yang Wang, Lei Li, and Markus Freitag. LLMRefine: Pinpointing and refining large language models via fine-grained actionable feedback. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 1429–1445, 2024a.

Wenda Xu, Guanglei Zhu, Xuandong Zhao, Liangming Pan, Lei Li, and William Yang Wang. Pride and prejudice: LLM amplifies self-bias in self-refinement. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15474–15492, 2024b.

Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step-by-step. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning (ICML)*, 2024.

A PROMPTS AND GENERATION HYPERPARAMETERS

Extracted verbatim from the pipeline notebooks that produced the checkpoints (`cross-llm-v2.ipynb` for seed 1234; `cross-model-critique-pipeline-v4-seeds-5678-9012.ipynb` for seeds 5678 and 9012). The two notebooks are byte-identical in every prompt template, the model loading and quantization code, and the generation call; they differ only in the seed list and an unrelated batch-size throughput setting.

Generation prompt (round 0 and all revisions, model A). System prompt:

```
You are a Python programmer.  Output only a complete Python
function in a ```python code block.  No explanation.
```

User prompt template (`{prompt}` is the HumanEval+ problem statement):

````
Complete this function.
```python
{prompt}```
````

The same template is reused for revision (below), substituting the current code and critique for the problem statement alone.

C1 and C2 critique prompt (neutral). Both conditions use the identical system prompt, differing only in which model receives it (model A for C1, model B for C2):

```
You are reviewing Python code.  List concrete bugs only.  If
the code is correct, reply exactly:  NO ISSUES FOUND.
```

C3 critique prompt (persona-injected). The full injected persona text, verbatim:

```
You are a senior software engineer with 15 years of
experience in safety-critical Python systems.  You are known
for catching subtle edge-case bugs that others miss:
off-by-one errors, empty-input handling, type coercion, and
boundary conditions.  You are rigorous and skeptical.  List
concrete bugs only.  If the code is genuinely correct, reply
exactly:  NO ISSUES FOUND.
```

Critique user prompt (shared by C1, C2, C3).

````
Specification:
```python
{prompt}```
Candidate implementation:
```python
{code}```
Review it.  Be specific:  name the input that breaks it.
````
````

Revision prompt (all conditions, model A). System prompt:

```
You revise Python code based on review feedback.  Output
only the corrected function in a ```python code block.
```

User prompt template, showing exactly how the critique string is inserted (`{critique}`):

```
Specification:
```python
{prompt}```
Current implementation:
```python
{code}```
Review feedback:
{critique}
If the feedback is wrong, keep the original code.  Output
the final function.
```

For C4, `{critique}` is substituted with the literal string `NO ISSUES FOUND` and no critic model is invoked; the revision call is otherwise identical to C1–C3.

Generation hyperparameters, as set in the notebook code. Temperature 0.2; `do_sample` is set to `TEMP > 0`, which evaluates to `True`; `top_p` $= 0.95$; maximum new tokens $= 640$. `top_k` is not set anywhere in either notebook; not set, library default. Neither notebook sets a repetition penalty or a beam count. The model's own bundled `generation_config.json` (part of the downloaded checkpoint, not present in this repository) governs any parameter neither the notebook nor this appendix names.

Quantization. Both models are loaded via `transformers.BitsAndBytesConfig` with:

- `load_in_4bit=True`
- `bnb_4bit_quant_type="nf4"` (NF4)
- `bnb_4bit_compute_dtype=torch.float16`
- `bnb_4bit_use_double_quant=True`

B Analysis assumptions

The stored checkpoints were ambiguous in several places. We list every decision we made, so that a reader can check whether any of them drives a result.

1. **Null `prev_passed` and `critic_said_clean` for C0 and for round 0.** The synthesized C0 records are copies of the round-0 record, which carries no `prev_passed` field and a null `critic_said_clean`. The original pipeline produced `None` for exactly these rows, and we reproduce that rather than backfilling. The consequence is that false-alarm rates are computed only over C1/C2/C3 rows at rounds ≥ 1, which matches the original run.
2. **Critique length is 0 for round 0 and for all C0 rows**, since the critique field is null there. This makes C0 indistinguishable from a zero-length critique in that column, which matters only for the length analysis in Appendix C, where C0 does not appear.
3. **Error types are parsed from the last non-blank line of the stored stderr tail.** The exception class name is taken as the leading identifier of that line. `AssertionError`, or any traceback containing a bare `assert` line, maps to `assertion`; `TimeoutError` or text matching a timeout pattern maps to `timeout`; a recognized class name outside our label set maps to `other`. Checked against all $1{,}872$ failing records in the merged dataset: no record has a blank error field, so the `empty` label is defined but unused, and `RecursionError` never occurs. Exactly 8 records fall to `other`: `AttributeError` (5), a NumPy ufunc loop error (2), and `UnboundLocalError` (1).

4. **Point estimates are pair-level; bootstrap intervals are problem-level.** The naive repair rate and the causal uplift are reported as pair-level proportions, in which each seed $\times$ problem instance counts once, while the bootstrap averages within each problem before resampling. These coincide exactly for the pooled subset, because the design is balanced at three seeds per problem; they differ slightly for the round-0-conditioned subsets, where a problem may fail in one seed and pass in another. We chose the cluster bootstrap to match the resampling procedure used in the original experiment's analysis and because the problem is the unit that repeats across seeds.
5. **"Seeds pooled" means clustering by problem, not treating the 492 pairs as independent.** Treating them as independent would understate interval width, since each problem appears three times.
6. **The harmful bucket for C1 is empty** ($n = 0$), so no box is drawn for it in Figure 3 rather than a degenerate box at zero.
7. **Round 1 only for causal claims.** Rounds 2 and 3 are not matched and are excluded from every causal quantity. They remain in the merged dataset and are used only for the descriptive final-round figures in Section 5.2, which are labeled as descriptive.

C CRITIQUE LENGTH BY BUCKET

Table 5 gives the full distribution summarized in Section 5.3, and Figure 3 plots it. The C1 rows are the direct evidence for the degeneracy documented in Section 6.1: every C1 critique is 15 or 16 characters long, which is the length of the null string with and without a trailing period.

Table 5: Critique length in characters by condition and counterfactual bucket, round 1, pooled over seeds. The C1 harmful bucket is empty. The A bucket contains 1–2 pairs and is not interpretable.

Cond.	Bucket	n	mean	median	q25	q75	min	max
C1	A `true_helpful`	1	16.0	16.0	16.0	16.0	16	16
C1	B `false_helpful`	406	15.7	16.0	15.0	16.0	15	16
C1	C `harmful`	0	—	—	—	—	—	—
C1	D `true_useless`	85	15.5	16.0	15.0	16.0	15	16
C2	A `true_helpful`	1	714.0	714.0	714.0	714.0	714	714
C2	B `false_helpful`	385	999.5	954.0	710.0	1243.0	246	2881
C2	C `harmful`	21	799.4	734.0	536.0	1146.0	389	1465
C2	D `true_useless`	85	1078.3	1042.0	734.0	1369.0	252	1977
C3	A `true_helpful`	2	907.0	907.0	829.5	984.5	752	1062
C3	B `false_helpful`	386	1088.0	1037.0	758.5	1323.0	305	3332
C3	C `harmful`	20	931.4	802.5	695.2	1084.0	432	2346
C3	D `true_useless`	84	1128.2	1106.5	740.0	1369.5	218	3052

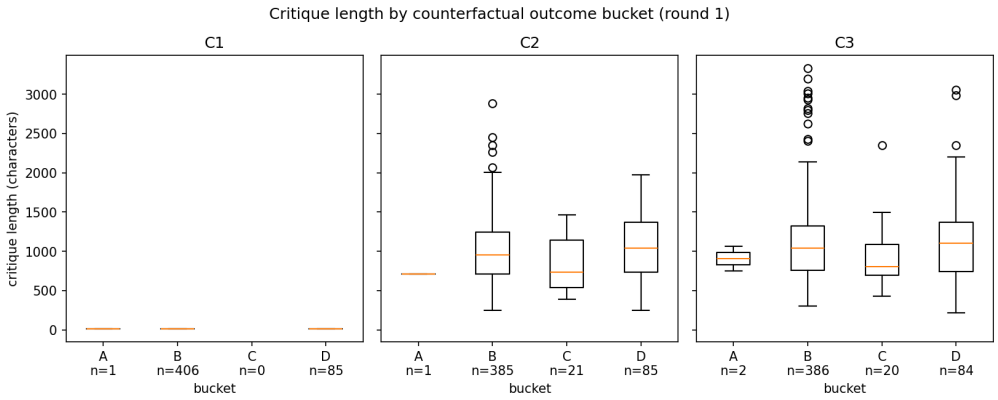


Figure 3: Critique length by counterfactual bucket, one panel per condition. The C1 panel is flat at 15–16 characters because every C1 critique is the null string. No box is drawn for the empty C1 harmful bucket.

D DATASET AND DERIVED TABLES

The merged trajectory table has 9,840 rows, one per (seed, condition, round, problem), and is unique on that key. It carries the pass indicator, the previous-round pass indicator, the regression/repair/churn flags, the critic's clean verdict, code length, critique length, and the parsed error type. From it we derive: the matched-pair table (1,476 rows, one per seed $\times$ advice condition $\times$ problem, each labeled A/B/C/D); the bucket counts of Table 1; the causal-effect table behind Tables 2 and 3; and the critique-length summary of Table 5. Every number in the paper is read from one of these four derived tables.

Two provenance details matter. The three seeds are drawn from two separate archives of the original run, and one of those archives contains a second, non-identical execution of seed 5678 — only 87/164 round-0 programs are byte-identical between the two, and their round-0 pass counts differ (133 against 135). We use the complete execution only and never mix them. The discarded execution is also incomplete, lacking a third round.

E MECHANISM CATEGORIES

Labels used in Section 5.3. *Hallucinated bug*: the critic asserts a defect the round-0 code does not have. *Spec misunderstanding*: the critique or revision misreads the problem statement. *Defensive logic*: added guards or type checks alter correct semantics. *Valid issue, wrong fix*: the critic identifies something real and the edit breaks it anyway. *Semantic drift*: the revision goes beyond what the critique asked for. *Broad rewrite regression*: a local remark triggers a disproportionate rewrite. *Unclear*: the evidence does not connect the critique to the edit.

Two categories were added because the evidence required them. *Critique text as code*: the critique's own prose was pasted into the program, and both instances yield `SyntaxError` because the critic emitted `NO ISSUES FOUND` inside a Python code fence. *Anchoring, no edit*: the critic affirmed a program that was in fact broken, the reviser made no change at all, and the program stayed broken while the null arm's unconstrained revision repaired it. Both instances are `HumanEval/129` under C3, in two different seeds.

Each case was assigned exactly one primary mechanism, so Table 4's counts sum to 41; several cases exhibited more than one of the categories above, and a released `secondary_mechanism` field records the second one for 29 of the 41 cases. The primary was chosen, case by case, as whichever category the reader judged to best characterize the causal chain from the critic's claim through the edit to the test failure — for instance, a case where an unfounded claim led to an overbroad guard is filed under *hallucinated bug* when the false claim is judged the dominant cause, or under *defensive logic* when the overbroad edit is. This is a single reader's judgment call on each case, not a scored or automated rule, and is exactly the reason Section 6.3 describes the resulting frequencies as descriptive rather than validated.

F BUCKET PROPORTIONS BY ROUND-0 OUTCOME

The visualization of Table 3, moved here for space. The underlying data is unchanged.

G DESCRIPTIVE ROUND-3 PASS@1

Moved here from Section 5.2 for space. At round 3, C2 reaches 0.774, 0.762 and 0.780 across seeds 1234, 5678 and 9012, and C3 reaches 0.732, 0.738 and 0.793, against 0.823 in all three seeds for the frozen C0 arm and 0.829, 0.823, 0.829 for C4. These figures move in the same direction as the round-1 causal estimates, but by round 3 the conditions have occupied different states for two rounds, so the comparison is not matched. We report them as descriptive only and make no causal claim from them.

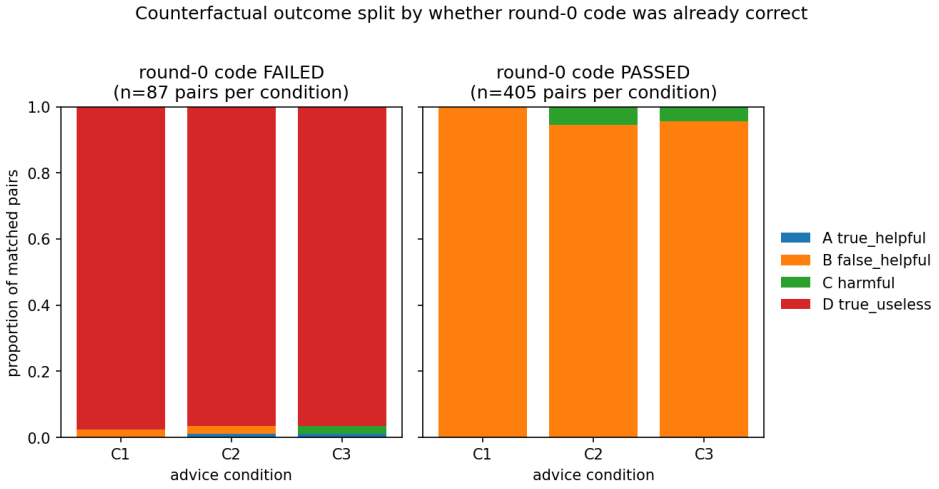


Figure 4: Bucket proportions split by whether the round-0 program passed. Left: the 87 failing instances per condition, almost entirely bucket D. Right: the 405 passing instances, where the harmful bucket appears for C2 and C3 but not C1.

H SANITY CHECKS

Four checks were run before any analysis, and all passed.

1. **Row count.** 9,840 rows, matching $3 \times 5 \times 4 \times 164$ exactly, unique on (seed, condition, round, problem).
2. **Round-0 pass@1.** $135/164 = 0.823$ in each of seeds 1234, 5678, and 9012.
3. **Round-0 byte identity.** $164/164$ programs identical across all five conditions, in every seed. This is asserted at load time and aborts the pipeline on failure.
4. **False-alarm reproduction.** This is a subset check against the original run, not the figure reported in the paper. Restricted to the two seeds for which the original run printed its own false-alarm figures, our merge reproduces them exactly: 0.000 for C1, 0.946 for C2, 0.989 for C3. The study figures quoted throughout the paper are instead the pooled three-seed rates over the full dataset, 0.000, 0.945, and 0.989.