PerfDiag: An Execution-Free Benchmark for Diagnosing Performance Regressions in AI Infrastructure

Anonymous authors
Paper under double-blind review

Abstract

Performance regressions in AI infrastructure can leave outputs unchanged while reducing throughput or increasing memory use and latency. Reproducing them often requires specialised hardware and workloads. We introduce **PerfDiag**: 430 (regression-inducing, fixing) commit pairs from four production repositories. Given the inducing commit, a model selects one of 32 mechanisms and cites a fix site, scored against annotations of the human fix. The release includes a read-only harness, decoys to bound false positives, and zero-call baselines for family, mechanism and fix site.

The best of six models identifies the cause family on about 58% of items. None of twenty prompt scaffolds significantly improves on a flat call for the one model tested; the sweep resolves differences of 10.2–17.9 accuracy points (§4). A model-authored, three-layer decision graph performs worse than the flat call on three of the five models we could run it on and on neither of the two weakest.

Claude Opus 5 wrote 94% of scored mechanism keys and ranks first. Re-scoring with a key it did not help write preserves the ranking, but label reproducibility does not establish correctness. All model scores use train/dev. No model run scores test, although an author inspected 11 test labels during the audit (Appendix I).

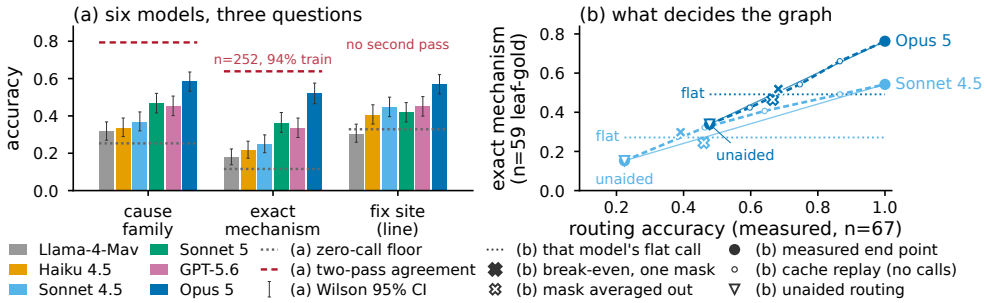

Figure 1: **(a)** Six models on PerfDiag's three scored outputs, using 344 items (311 for exact mechanism). Error bars are Wilson 95% intervals. Dotted lines show the strongest zero-call baseline for each metric. Dashed lines show two-pass agreement for family and mechanism on 252 items with both labels, including 238 train items. Fix-site scores use raw coordinates; there is no agreement line because the second pass did not cite sites. **(b)** Exact-mechanism accuracy versus measured family-routing accuracy as the true family is supplied for increasing fractions of items. Only the endpoints are measured; 402 interior item-runs reuse cached endpoint answers. Filled and open crosses are two estimates of each model's break-even point. Both models fall below their thresholds. The ordering of the shortfalls depends on the injection mask. Appendix G explains the two denominators used here and gives crossings on the matched axis used in §5.

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.

1 Introduction

A performance regression can pass functional tests while reducing throughput or increasing memory use or latency. Diagnosing it requires understanding how a code change affects execution. PerfDiag evaluates this diagnosis using the explanations and code changes in subsequent fixes (Figure 2).

Execution-based benchmarks must reproduce the relevant hardware and workload. Distributed training and inference can require multi-GPU H100 or GB200 nodes and a harness absent from the repository, restricting benchmarks to reproducible cases (He et al., 2025; Shetty et al., 2025; Ma et al., 2025; Nangia et al., 2026). PerfDiag instead evaluates diagnosis from the inducing commit against the human fix, without executing code. Scores measure agreement with the key, not repair success or developer usefulness (§8).

We make three contributions:

- **An execution-free benchmark** (§2). PerfDiag contains 430 commit pairs from vLLM (Kwon et al., 2023), Megatron-LM (Shoeybi et al., 2019), DeepSpeed (Rajbhandari et al., 2020) and TransformerEngine (NVIDIA, 2025), labelled with 32 mechanisms in 11 families. It includes a read-only harness and 3,642 (decoy, item) negatives in three strata whose contrasts are unpaired by construction. We score 471 decoy commits on 3 of the six models (Appendix G). Table 1 and Figure 1a report two-pass label agreement and the strongest of 9 zero-call baselines for each scored output. **(C1)**
- **No significant prompt improvement** (§4). We test 20 variants on Claude Sonnet 4.5 and 67 items. None significantly outperforms one call with all 32 labels at the declared cap (§3). 4 variants perform significantly worse; two variants have higher exact-mechanism point estimates (19/59 and 18/59 versus 16/59). Adding the commit message leads on dev; a cap-matched rerun on 344 items improves one of three models nominally, but not after correction, and reduces accuracy on both Claude models. Across prompts on Claude Sonnet 4.5, a permutation test rejects a shared cause-family accuracy ($p = 0.0043$), but neither draw rejects a shared fix-site accuracy ($p = 0.52$). The latter test has insufficient resolution to establish an absence of prompt effects (§4.1). **(C2, C5)**
- **Routing limits a diagnostic graph** (§5). Walking a model-authored, three-layer graph performs worse than the flat menu on three of the five models we could run it on and on neither of the two weakest. Supplying the correct family instead improves exact-mechanism accuracy by 2.00× [1.444, 3.111] on Sonnet 4.5 and 1.55× [1.212, 2.087] on Opus 5; neither gain is significantly above 1.5× after correction. Break-even routing accuracies are 0.469 [0.386, 0.582] and 0.664 [0.627, 0.712] over mask redraws. Graph routing falls well below both. Sonnet 4.5's flat prediction falls 0.029 below its threshold when averaged over uniform masks, but above it under the sweep's single mask. Only 22% of redraws preserve the ordering of the two shortfalls, so the data do not establish that routing deficits grow with model capability. **(C3, C4)**

2 PerfDiag

Items. Each item pairs commits $(c_{\text{ind}}, c_{\text{fix}})$ from one repository, where c_{ind} introduces a regression and c_{fix} repairs it. The model receives the inducing diff, changed-file list, repository name and author date. Diffs longer than 900 lines are truncated and marked; 24% of scored items are truncated. The highest per-model fix-site accuracy is 0.296 on truncated items and 0.659 on the rest. Commit messages are withheld by default because they can state the mechanism directly; §4 compares conditions with and without them. The fix commit is always withheld.

Item collection and attribution. We consider 622 candidate pairs and record how each inducing-to-fixing link was established. In tier A (164 items), the upstream project identifies the inducing SHA through a reference, revert, maintainer bisection or CI result. In tier B (266 items, 62% of the benchmark), we infer the link, usually with SZZ. SZZ attributes lines changed by a fix to the commit that last modified them (Śliwerski et al., 2005); its precision against developer-informed references is limited (da Costa et al., 2017; Rosa et al., 2021). Unresolved tier C pairs are excluded.

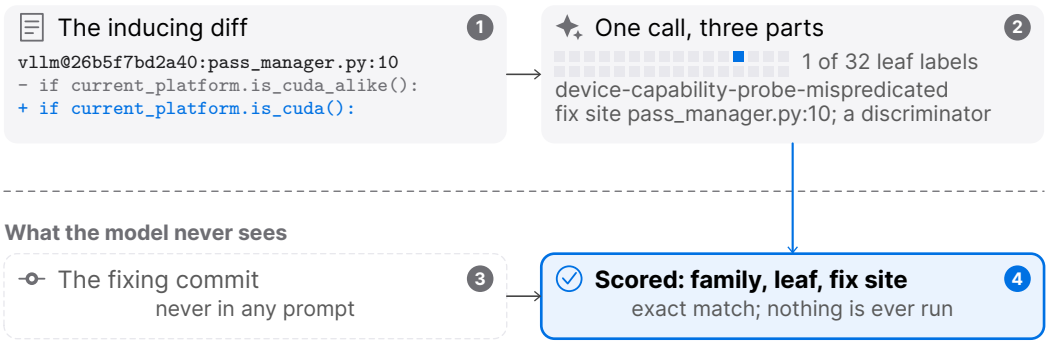

Figure 2: One PerfDiag item. 1 The model receives the inducing diff, changed-file list, repository and date; the commit message is withheld. The highlighted change reduces throughput. 2 In one call, the model returns a mechanism from 32 labels, a fix site and a discriminator. 3 The human fix supplies the evaluation reference and is never shown to the model. The dashed boundary and hollow fixing-commit card mark withheld information. 4 We score mechanism at the leaf and family levels and fix site at the line level. The discriminator is checked for format only. No code is built or executed. The displayed hunk reproduces two changed lines from one train item.

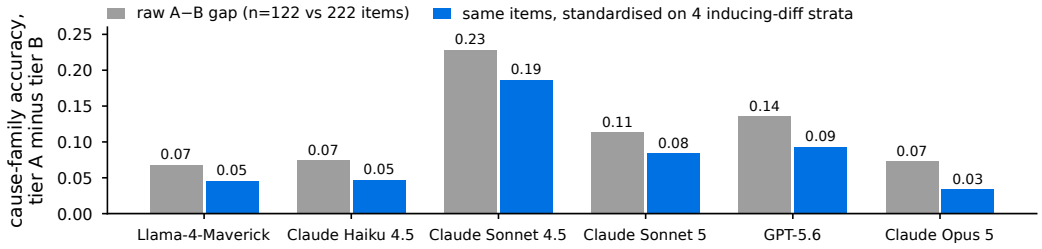

Figure 3: Cause-family accuracy gaps between tier A (upstream-established links) and tier B (author-inferred links, mostly SZZ, covering 62% of the benchmark). Grey bars show raw gaps; blue bars standardise for inducing-diff size. Standardisation reduces every gap without eliminating it. Fix-site gaps, omitted here, are larger and significant for all models. These bars have no intervals, so their direction is more reliable than their magnitude. Remaining gaps may reflect both attribution errors and difficulty not captured by diff size. The label audit assesses labels; §2 separately describes the link audit.

All 6 models score better on tier A than tier B. Fix-site gaps range from 0.165 to 0.247 and are significant for all 6 models, compared with 2 significant cause-family gaps (Figure 3). A blind, model-adjudicated audit covers the 153 SZZ-attributed links among 222 scored tier-B items. Its corrected precision is 0.980 [0.930, 1.000]; the upper endpoint is clipped, and both controls bias the estimate upwards. The most stringent adjudicator alone gives 0.814. The other 69 tier-B links were inferred by reading both diffs and have not been audited. They score below the SZZ links on 5 of 6 models, so the audit estimate should not be applied to them.

The strongest model achieves 0.631 on 122 tier-A items and 0.559 on 222 tier-B items (Appendix F). Appendix A describes the collection funnel and its two population filters. The larger filter is Claude Opus 5's inability to place 197 of 622 cards in the taxonomy; the annotation panel subsequently admits 96 of them.

Task. An answer has three outputs. We score two separately, without assuming they measure distinct abilities (§4.1). **M1**, the mechanism, selects one of 32 leaves and receives credit at the exact-leaf and 11-way family levels (Table 19; Appendix I describes looser conventions). For 33 items, the gold label specifies only a family. These items contribute to family accuracy but not exact-mechanism accuracy, yielding different denominators (Table 1). The taxonomy covers AI-infrastructure regressions rather than performance bugs in general.

M2, the fix site, lists (file, line, symbol) triples. A prediction receives line-level credit if any cited line is within ±12 lines of a line changed by the fix, a 25-line window. Gold lines use the fix commit's parent coordinates; predictions refer to the inducing diff. We report the original, *raw* comparison and a remapped comparison in Appendix G.4. **M3**, the discriminator, states a prediction that would support or refute the diagnosis. We check its format but never score its correctness.

Of 187 recorded runs, 4 use repository tools. They record 0 leak attempts across 1,605 tool calls. All tool access resolves against a read-only view pinned to c_{ind}'s parent (Appendix A).

Label provenance. Claude Opus 5 wrote a mechanism label, or *card*, for 334 of the 430 released items using prompt `annotate_label.v1`. Thus, the model ranked first in Table 1 wrote the answer key for 324 of the 344 scored items (Appendix I distinguishes the released and scored pools). A three-model panel labelled 129 additional eligible pairs without cards. It reached a majority on 96; the remaining 33 contested pairs are excluded. The panellists, Claude Opus 5, GPT-5.6 and Claude Sonnet 4.5, are 3 of the 6 evaluated models. They can therefore receive credit for matching labels they helped write, a concern given evidence of model self-preference under explicit rubrics (Wataoka et al., 2024; Pombal et al., 2026). Entry to the panel pool also requires two annotators to agree, excluding the hardest unresolved cases. We report the two label sources separately (Appendix E).

A second labelling pass supplies the support list under each taxonomy leaf. On 252 scored items with both labels, its agreement with the cards is 0.639 for mechanism and 0.794 for family (κ = 0.626 and 0.762). Both passes use Claude Opus 5, so this measures within-model consistency. A key constructed solely from the other annotators' votes preserves the ranking in Table 1, with Claude Opus 5 first at 0.644 on 275 items. That accuracy cannot be compared directly with Table 1, because the 69 excluded items are harder (Appendix J).

The reported *two-pass agreement* is not an accuracy bound. It is measured on easier items: the strongest model scores 0.665 where the passes agree and 0.423 where they disagree, with the same ordering for all 6 of 6 models (Table 20). Other annotators, who can read the withheld fix diff, match the existing key by majority on 0.775 of 315 items. Their extra information prevents interpreting this agreement as a ceiling.

To assess correctness, an author inspected fix commits for a pre-specified stratified sample of 19 scored items, accepting 19 mechanisms and rejecting 0. The 95% lower bound is 0.832, excluding label error rates above about 17% but not smaller rates. Coverage is uneven: 15 sampled items have corpus labels and only 4 have panel labels. The panel-specific exact lower bound is 0.398, allowing error rates up to 60% (Appendix I).

Coverage measured on the items the taxonomy was induced from would be circular; the non-circular measurement is that on the 129 items with no card label, 3 models from 2 vendors took an explicit "none of these fits" option on 0.000 to 0.039 of the items each answered, rising to 0.097 on a sampled subset once the family the annotator itself chose was deleted from the menu, against 31% unplaceable with no menu at all (Appendix I).

Zero-call baselines. Table 31 reports 9 baselines on the same items and scorer as the models; Table 1 uses the strongest baseline in each column. For cause family, a frozen microsoft/unixcoder-base reaches 0.253, statistically indistinguishable from the character-n-gram classifier at 0.238 (p = 0.66). None of these baselines is fine-tuned, and the largest uses a 126 M-parameter encoder. The comparisons therefore apply to these implementations, not all possible zero-call predictors. Only 4 of 6 models significantly outperform the best family baseline after whole-paper correction. Llama-4-Maverick does not do so even before correction (p = 0.057); Claude Haiku 4.5 does so nominally (p = 0.012 against a threshold of 0.01, so we report the direction and not a rejection). For exact mechanism, the character classifier reaches 0.116 against a majority-label rate of 0.029. For fix site, citing the three largest hunks reaches 0.328.

Skew and uneven difficulty. The corpus is repository-skewed: 212 of the 344 evaluated items, or 62%, are vLLM, so every headline number is weighted towards one project's engineering style (Table 12). Difficulty is uneven too: over the 11 families Opus 5's cause-family accuracy runs from 0.355 to 0.733, so a headline number hides which branch a model fails on (Appendix E).

Table 1: Six models on 344 items, using one flat call with 32 labels and no commit message. Identifiers are in Table 3; only 2 of 6 are dated snapshots. † marks 3 label authors; Claude Opus 5 wrote 324 keys. *Cause*: 11-way family, interval to two decimals. *Mechanism*: exact leaf. *Both*: mechanism and fix site. *Diff withheld*: closed-book control. *Fix site*: a citation within ±12 lines of the fix, in *raw* fix-parent or *mapped* prediction coordinates; both are lower bounds (Appendix G.4). Family/mechanism denominators are 344/311; unparseable answers count as wrong. 24 parsed answers from Claude Sonnet 5 and Claude Haiku 4.5 encode `sites` as strings and score zero. Imputing each model's well-formed rate gives 0.439 and 0.417 raw accuracy (Appendix D). Dashes denote unrun or undefined cells; *Diff withheld* covers 2 models. The *Both* baseline is unmeasured; the closed-book baseline is 0.331 (§6). Other baseline cells use the strongest zero-call arm (Appendix G.5). *Two-pass agreement* compares one annotator's labelling passes on 252 shared items (238 train, 14 dev, no test), drawn from the taxonomy-induction pool. It is not an accuracy ceiling.

	Diff shown						Diff withheld
				Fix site			
Model	Cause	95% CI	Mechanism	raw	mapped	Both	Cause
Llama-4-Maverick	0.317	[0.27, 0.37]	0.177	0.305	0.392	0.080	–
Claude Haiku 4.5	0.337	[0.29, 0.39]	0.215	0.407	0.529	0.141	–
Claude Sonnet 4.5†	0.369	[0.32, 0.42]	0.248	0.448	0.558	0.161	0.262
Claude Sonnet 5	0.468	[0.42, 0.52]	0.363	0.419	0.552	0.196	–
GPT-5.6†	0.453	[0.40, 0.51]	0.334	0.451	0.581	0.235	–
Claude Opus 5†	0.584	[0.53, 0.64]	0.521	0.570	0.701	0.376	0.340
zero-call floor	0.253	–	0.116	0.328	0.413	–	–
two-pass agreement	0.794	–	0.639	–	–	–	–

Splits. Items are ordered by fix date into 258 train, 86 dev and 86 test items. All reported model scores use train or dev; the results database rejects test-split runs. The test key is not entirely unseen: a sampling error in the human label audit exposed 11 of 86 test labels to an author (13%, all 11 panel-labelled; §8). The prompt search in §4 and all dev-only paired tests use an earlier 67-item dev set. Confirmation runs combine train with the current 86-item dev set.

Test composition also differs from the scored pool. Panel labels account for 76 of 86 test items (88%), versus 6% of 344 scored items. Test contains 87% vLLM items and 49% tier-A items, versus 35% tier-A items in train/dev (Table 9). A dev-to-test change therefore combines shifts in provenance, repository mix and attribution tier; the tier shift favours easier items. Panel items satisfy all six mechanical filters (Table 4, Appendix A). Finally, splitting ignores shared inputs: 8 test items duplicate dev inputs, including 2 with matching labels. Shared inputs limit scored-pool accuracy to 0.956 for family and 0.952 for mechanism (Appendix F).

3 Setup

We evaluate six models from three vendors (Table 1); Table 3 gives their identifiers. We exclude DeepSeek-R1 because the platform rejects its tool-use requests with HTTP 400. This prevents harness execution and is not an evaluation result. We request a 32,000-token output cap, reduced to 8,192 for one model because of a lower platform limit; the reduced cap is non-binding. Of 136 live cells, 81 record the requested cap, 50 omit it and are assigned the harness default of 4,096, and 5 record a lower value. Output limits affect the scores: at 4,096 tokens, the two strongest models truncated 50% to 52% of calls. This invalidated the Sonnet 5 cell and made Opus 5's score a lower bound (Appendix G).

One evaluation covers 344 items, with one call per item and no local GPU. Two-arm comparisons are paired by item, using exact McNemar tests and paired bootstrap intervals; individual proportions use Wilson intervals. We report invalid cells rather than discarding them.

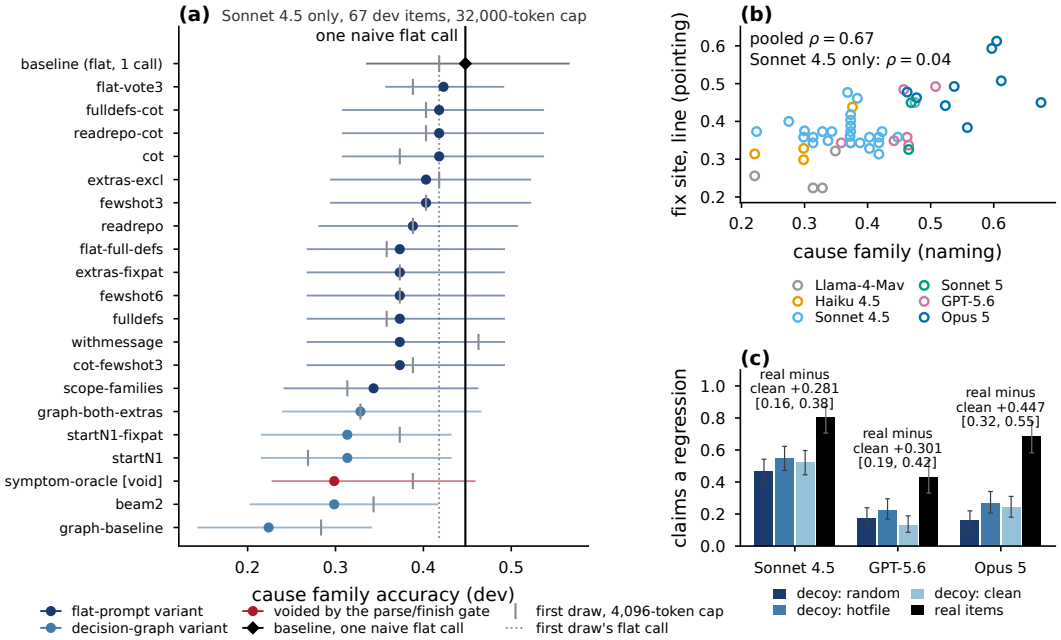

Figure 4: **(a)** All variants on Claude Sonnet 4.5 and 67 dev items at the 32,000-token cap, sorted by family accuracy. The two flat-call cells share a row. Bars show Wilson intervals; red marks an invalid cell and the black line marks the flat baseline. Table 26 defines the codenames. Grey ticks show the first draw at 4,096 tokens, with its baseline dotted. This sweep covers one model. **(b)** Family versus fix-site accuracy across configurations, coloured by model. The pooled correlation ($\rho = 0.667$) reflects between-model differences. Sonnet 4.5's 27 configurations give $\rho = 0.040$, with uncertain sign (§4.1). Closed-book cells are excluded because their lack of a diff makes site citation impossible. **(c)** Regression detection under neutral framing: three decoy strata (163–171 commits) and 86 real items (black), with Wilson 95% intervals. Numbers above each model give recall minus its R-CLEAN rate, with Newcombe 95% intervals. The intervals overlap, so the displayed ordering relies on point estimates (Appendix G).

4 Prompt scaffolds do not significantly improve on a flat call

We test 20 variants on Claude Sonnet 4.5 and 67 items (Figure 4a). None significantly outperforms one flat call. This result is limited to one model and a small evaluation set. Raising the output cap and repeating the calls changes cause-family accuracy by 0 to 6 items. That range contains 16 of 19 comparable variants; all 3 outside it perform worse than the flat call.

The variants include chain-of-thought that first describes the extra work introduced by the commit (Wei et al., 2022); three- and six-shot train demonstrations (Brown et al., 2020); repository access at c_{ind}'s parent with up to six tool turns (Yao et al., 2022); full label definitions; three-sample self-consistency voting (Wang et al., 2022); and 6 decision-graph variants (§5). The remaining 14 vary the flat prompt.

Adding the commit message gives the largest dev gain in the first draw at 4,096 tokens: +4.48 points for family and +11.86 for mechanism on Sonnet 4.5. The family gain is not significant (13 discordant items, exact McNemar $p = 0.58$). The mechanism comparison favours the message arm by 7–0 discordant outcomes (7 total, $p = 0.016$), which does not clear the corrected threshold of 0.011 (Appendix G).

A cap-matched rerun on 344 items gives smaller, model-dependent changes in exact-mechanism accuracy: −2.25 points for Sonnet 4.5 ($p = 0.39$), −1.93 for Opus 5 ($p = 0.38$), and +5.14 for GPT-5.6 ($p = 0.026$). GPT-5.6 clears the pre-registered per-comparison threshold, but it does not survive Benjamini-Hochberg across the 186 corrected tests (rank 50 of 186, threshold 0.013). The

pre-registration omitted a multiplicity correction; we therefore report this gain as a point estimate that does not survive correction.

We also split messages by whether they contain performance or repair vocabulary. A permutation test does not distinguish the two groups, so the data do not rule out gains from explicit problem descriptions in the input. Outside the selection pool, the message effect changes sign across vendors and the earlier leakage-check result reverses. Appendix G reports both analyses and their intervals. It also shows that trained routers perform below every evaluated model, while removing the 32-label menu reduces accuracy and invalidates the cell.

The prompt sweep has limited statistical resolution. Across 40 (variant, metric) cells, the smallest detectable paired difference ranges from 10.2 to 17.9 points, with a median of 13.5; 2 cells have no power. Failure to detect an improvement therefore does not establish that prompting cannot help. Appendix N gives this analysis and three further properties.

4.1 Prompt effects on diagnosis and localisation

Cause-family and fix-site accuracy correlate across models, but the within-model relationship remains unresolved. Pooling 52 configurations gives $\rho = 0.667$, reflecting differences in model capability. Across 27 prompts for Sonnet 4.5, the declared-cap draw gives $\rho = 0.040$ with bootstrap interval $[-0.348, 0.375]$ (Figure 4b); the harness-default draw gives 0.039. The uncertain sign supports neither independence nor a stable association (Appendix H).

Prompt variation has a larger estimated effect on naming: the cause-family signal SD is 0.038, whereas the fix-site signal SD has a point estimate of zero and a 95% upper bound of 0.019. This difference holds at all 4 site granularities computed by the scorer (Appendix G.4). Localisation alone therefore cannot be assumed to proxy diagnosis, even though it provides an execution-free evaluation target used by the closest prior work (Yang et al., 2026).

Fix-site comparisons also depend on the citation budget. The heuristic cites three sites, while Llama-4-Maverick averages 1.40. Limited to one site, the heuristic scores 0.206 and the model significantly outperforms it ($p < 0.001$). At three sites, the heuristic scores 0.328 and the difference is not significant ($p = 0.38$, remapped $p = 0.56$). The other 5 models outperform it in both coordinate frames, by up to 0.241. Thus, the largest-hunk heuristic matches the lowest-ranked model only when given a larger citation budget. The 3 other site criteria produce different rankings; at file level, the order matches cause-family accuracy (Appendix G.4). Across the six models, cited-site precision ranges from 0.604 to 0.686, with 1.26 to 2.91 cited files. The scorer computes precision, but the original aggregation omitted it.

5 Routing errors limit the diagnostic decision graph

We evaluate a three-layer graph that selects an observed symptom, then a cause family, then a mechanism. The model commits to each layer before moving to the next. Claude Opus 5 authored the graph from train items under prompt `node_author.v1`, which also produced the 32-label set. The same model produced the item keys under `annotate_label.v1`; this experiment therefore evaluates its proposed structure.

Relative to the flat call, graph traversal reduces cause-family accuracy by 10.45 to 22.39 points for 3 of 5 models with valid graph cells (Figure 5). We detect no reduction for the 2 weakest models. These rows use different output caps. Under matched caps, Sonnet 4.5 still leads the strongest model by 2.99 points (Appendices H and G).

Supplying the correct family improves exact-mechanism accuracy over the flat menu by 2.00× on Sonnet 4.5 and 1.55× on Opus 5 (Figure 7), paired bootstrap [1.444, 3.111] and [1.212, 2.087], neither clear of 1.5 after the correction. Sonnet 4.5's flat-call accuracy is below the within-family chance rate of 0.294, which partly explains the larger ratio. All gains occur on items that the flat call assigns to the wrong family, as the comparison requires by construction. On correctly routed items, restricting the menu yields no detectable change.

To control for presentation, we rerun the oracle arm without per-edge conditions, matching the text emitted by the two arms. The rerun shows that neither multiple falls once that text is removed; the

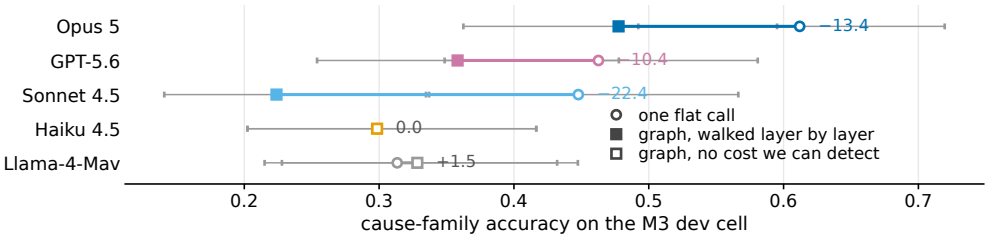

Figure 5: Graph and flat-call results on each model's `M3` dev cell. Circles denote flat calls; squares denote graph traversal, filled for detectable deficits. Numbers are graph-minus-flat accuracy points. Rows follow model capability, which does not monotonically predict the deficit. Caps differ: Llama-4-Maverick, Claude Haiku 4.5 and GPT-5.6 use 4,096; Claude Sonnet 4.5 and Claude Opus 5 use 32,000. Table 18 gives matched-cap results, including Claude Sonnet 5, absent here. Grey bars are Wilson 95% intervals. In item-paired exact McNemar tests, only Sonnet 4.5's deficit survives correction ($p = 0.0015$ versus 0.0075; Appendix H).

two estimates are statistically indistinguishable (Appendix G). We report the matched arm, which still combines the true family label with a shorter menu.

The overall routing deficits relative to the flat menu are 0.203 and 0.186 over 59 items. These are the largest contrasts for both models. The Sonnet 4.5 deficit survives whole-paper correction ($p = 0.0068$ versus 0.0097); the Opus 5 deficit does not ($p = 0.02$ versus 0.012). No individual component of the deficit is significant for either model. Presenting all three levels in one call, without committing at intermediate layers, still reduces exact-mechanism accuracy by 0.068 and 0.034 ($p = 0.29$, $p = 0.72$). Skipping the symptom layer changes routing by intervals that include zero. Appendix G reports all 24 contrasts among the four arms.

Layer-level scores need careful interpretation. The symptom score accepts a set of classes: a gold mechanism is reachable from an average of 3.41 out of 6 symptom classes. Always answering `throughput` scores 0.966, above every model (Appendix H).

Injecting the true family for increasing fractions of items yields break-even routing accuracies of 0.469 and 0.664 on 59 items with leaf labels (Figure 8). Graph routing falls well below these thresholds. Reading the family from a flat prediction gives 0.441 and 0.661, close to the thresholds without clearly exceeding them under uniform masks. The thresholds are closed-form functions of four proportions, rather than fitted sweep curves. They are sensitive to individual items, and the ordering of the two shortfalls reverses across estimators. The robust result is the gap between each threshold and the graph's own routing accuracy: no mask closes it. Appendix H gives paired intervals and resample counts, and Appendix G derives the thresholds.

For Sonnet 4.5, a ten-point routing gain corresponds to about 5.02 points of exact-mechanism accuracy. This slope comes from interpolation between two measured endpoints. Improving this design therefore requires a 11-way router that outperforms the model's flat family prediction. Our trained routers do not meet that requirement, including hierarchical routers using the inference procedure of Pan et al. (2023) (Appendix G).

6 Memorisation, detection and label circularity

We examine three threats to the interpretation of Table 1.

Memorisation. Removing the diff leaves repository, date and symptom class. Opus 5 drops from 0.584 to 0.340 (gap 0.244). Sonnet 4.5's smaller gap of 0.108 does not establish memorisation: its closed-book score does not differ significantly from the zero-call baseline (paired $p = 0.55$). Symptom labels alone yield 0.331 without model calls. Withholding them reduces the models to 0.198 and 0.128, neither significantly above the majority-family rate of 0.166 ($p = 0.17$, $p = 0.11$). The closed-book score therefore largely reflects symptom information (Appendix O).

Detection versus diagnosis. Under neutral framing, 3 models flag 0.129–0.521 of 163 clean decoys, with recall of 0.430–0.802 on real regressions (Figure 4c). False-positive rate rises with recall

(ρ = 1.00 on R-CLEAN, chance probability one in six; 0.50 on R-RANDOM), so it alone cannot rank discrimination. Recall minus false-positive rate is 0.281–0.447. Newcombe intervals span 0.159–0.553 overall, overlap, and each exclude zero. Since decoys are unvalidated negatives, false-positive rates are upper bounds and these differences are lower bounds (§8). The models show some discrimination, with uncertain magnitude. PerfDiag evaluates diagnosis given a regression; Appendix G reports all 9 detection cells.

Label circularity. 3 evaluated models helped write the panel labels (§2). Several checks assess the sensitivity to this overlap. Reproducibility differs by up to 18.50 points between pools of 51 and 252 items. Claude Opus 5 is not ranked first on the 7 items for which it cast no panel vote, or on the 19 non-blind, human-checked scored items, most of which carry its labels (Table 38). An external comparison with 2 human-authored performance-bug taxonomies finds no prior analogue for 14 of our 32 leaves. Conversely, 6 of their 11 root-cause categories have no corresponding leaf, or 2 under a looser mapping (Appendix I).

6.1 Reporting requirements

A PerfDiag report should include seven elements: separate cause-family, exact-mechanism and raw line-level fix-site scores with the corresponding baselines; answer format; label provenance and repository composition (Table 11, Figure 6b); output cap and both truncation rates; token cost alongside accuracy (Figure 6a, with a point-estimate Pareto front of depth 4); task framing; and accuracy by self-reported confidence. Appendix L evaluates each element.

7 Related work

Performance-improvement benchmarks evaluate models by executing code, without a separate diagnosis score or a mechanism label set (He et al., 2025; Shetty et al., 2025; Ma et al., 2025; Nangia et al., 2026). EffiHolmes (Yang et al., 2026), the closest benchmark using human fixes as a reference, measures fix-location accuracy on 140 Python issues. It executes repositories and does not provide a mechanism taxonomy. Other work evaluates mechanism identification from inputs other than the introducing commit (Chen et al., 2025a; Sijwali et al., 2025; Li & Wang, 2026; Gu et al., 2026). Appendix M reviews these ten studies and the dataset-construction methods we draw on (Kim et al., 2021; Pan et al., 2023).

To our knowledge, PerfDiag is the first benchmark to jointly identify a performance-regression mechanism from the introducing commit and cite the site that needs to change. We do not claim execution-free evaluation itself as novel (Egersdoerfer et al., 2025). Comparisons using Table 1 should account for the reporting factors in §6.1.

8 Limitations

The human audit gives source-specific correctness bounds: 15 corpus-labelled items give a Clopper–Pearson lower bound of 0.782, while 4 panel-labelled items give 0.398, allowing panel error rates up to 60%. This uncertainty particularly affects test, where panel labels predominate (§2). An author also inspected the fix diff and label of 11 test items (13%); all were accepted, giving a lower bound of 0.715 on that subset. No reported model accuracy uses these items, but their inspection reduces the independence of future test evaluations (Appendix I).

Decoy false-positive rates are upper bounds because the negatives are unvalidated. Transfer from these four Python-dominant repositories to compiled kernels is untested, as is the connection between diagnosis accuracy and useful repairs or review comments. Each cell uses one run.

The link audit uses 3 model adjudicators, is biased upwards, and lacks a record of which model originally inferred the links. Its harder negative control, blame's runner-up, was not evaluated; it accepted 13 of 16 links that blame never reaches. The 19-item label check was not blind, included no planted errors and found none. Only 2 of 5 adjacent fix-site rankings differ significantly. The 126 M-parameter baseline does not bound stronger zero-call alternatives. Identifiability from the inducing diff is unmeasured, so low accuracy does not establish headroom.

Reproducibility statement

Every number in this paper is generated from files in the supplementary material. The harness reaches the repository only through a read-only view pinned at c_{ind}'s parent and takes no argument through which a model could name a revision (Appendix A). Model calls go through one API behind an on-disk cache keyed on the exact request, temperature 0 where the model accepts it; the requested output cap is clamped to 8,192 for the one model with a lower platform limit, where it is non-binding, and Appendix C defines *arm*, *variant* and *cell* and the parse/finish gate that voids a cell. Appendix A gives the provider identifiers, the exact answer schema, the cache key and the reconstructed window in which the calls were made; `code/rebuild_all.sh` runs the whole chain from the run files to this PDF in our checkout. Inside the extracted archive it does not, and a few tables there ship as generated rather than rewritten because their generators need an input the archive does not carry. The exit code, the last step it reaches and the list of those tables are measured when the archive is built and recorded in the `MANIFEST.json` written beside it.

Limits to exact reproduction. Three parts of the process cannot be reproduced exactly. First, the labelling pass ran Claude Opus 5 over 449 cards at reasoning effort `medium`, without setting temperature, top-p or seed. The corpus release notes state that this pass is not byte-reproducible. Repeating it with the scored-call configuration could change `node_id` labels and hence Table 1. Because the label author also ranks first, we assess this dependence by re-scoring against independently constructed keys (§6, Appendix J). The released labels are sufficient to reproduce the reported scores without repeating annotation.

Second, four of the six evaluated provider identifiers are undated aliases; only two name dated snapshots (Table 3). The aliases include Claude Opus 5, which both labels the data and leads Table 1. Later calls to these aliases may use different weights, so they support comparisons within a model family rather than exact replication. Third, released run rows omit the request temperature, preventing reconstruction of the full cache key (Appendix A).

The second labelling pass. Its prompt was not saved in a versioned file. The archive therefore cannot verify the paper's description of that prompt or the claimed distinction between the two passes. The agreement itself is reproducible: `probe/ceiling.py`, using `graph/graph.json` and `labels.jsonl`, returns 0.794 and 0.639 (Appendix I).

Release, versioning and maintenance. The submitted supplementary archive is the corpus: the 430 item records, the four repository pins in `corpus/repos.lock`, the label graph, every per-item run file and the append-only run database. It is self-contained and carries no repository contents, so nothing in it can rot when an upstream project rewrites history; a user who wants the diffs runs `code/clone_upstream.sh`, which fetches the four repositories at exactly those pins. We will deposit the same archive under a DOI at camera-ready time and cannot name the host here without breaking anonymity. Versioning is by the stamps the archive already carries: `graph_version` on every run row and `stamp_version` on the splits, so a number is comparable to another only when both stamps match. That comparison is on the reader: the aggregation guard we do enforce refuses to pool cells that do not cover the same items, and nothing in the harness compares stamps for you. Corrections we expect to have to make are label corrections; the run database is append-only and supersession is recorded rather than deleted, so a corrected label leaves the superseded row and its reason in place.

Ethics statement

The corpus is built from public commit histories and carries no personal data beyond what those histories already expose: we release commit identifiers, labels and splits, and neither the release nor any prompt contains a commit author's name or address. The four upstream repositories are read at the commits `corpus/repos.lock` pins and stay under their own licences, Apache-2.0 for three of them and BSD-3-Clause for the fourth; we release the corpus under CC BY 4.0 and the harness, scorer and generators under the MIT licence, which leaves the upstream attribution obligation in place. The answer key is model-authored, so what a score here certifies is agreement with a machine-written naming convention and not diagnostic correctness (§2). A high score should not be treated as

evidence that an automated regression triager is ready for deployment, given the false-positive rates in §6. Appendix O gives the provenance, the per-repository licence reading at each pin, the scope of the conclusions and the misuse argument in full.

AI USE STATEMENT

We evaluate language models and also use them in data construction, analysis and writing.

As the subject. The 6 models in Table 1 are the subjects of the evaluation; the uses described below concern the research process.

Generating and cleaning the data. The 32-label taxonomy and the mechanism label on each item were authored by a language model, and the 96 panel labels come from a majority vote of three models. §2 and Appendix I report the measured reproducibility of those labels, the human spot check, and which three of the annotator models are also evaluated models. The 2,941 decoy commits were assembled by script from commit metadata, not generated by a model.

Auditing the data with models. The blind audit of the (inducing, fixing) link in Appendix F put 395 worksheet rows to 3 model adjudicators plus one cross-vendor model, a total of 1,580 model calls, and its verdicts are the majority of those models and not of people. Its corrected reading 0.980 and its conservative reading 0.814 are therefore model judgements about real commits; they are not a manual audit, and they are not pooled with the 19-item human record of Appendix I. No adjudicator in the majority is one of the 6 evaluated models. No item, diff, or fix commit is synthetic: every one is a real commit in a public repository.

Implementing and running the experiments. The harness, the scoring code, the table and figure generators and the mechanical gates were written with a coding assistant, reviewed line by line by the authors, and are released in the supplementary material. The authors are responsible for every line, including the eight instrumentation failures, in three classes, disclosed in Appendix G.

Analysing and writing. A language model was used to draft and edit prose, to propose figure layouts, and as one of several adversarial readers during revision. Every number in the paper is emitted by the scripts in the supplementary material from the append-only run database; no number was produced or transcribed by a model. Most bibliographic entries were retrieved from publisher APIs by `code/fetch_bib.py` and checked against the publisher record, and `results/bib_raw.json` caches what those queries returned. The entries `references.bib` marks as carried over are the exception: for those the script keeps the text the file already has rather than rebuilding the entry from a query. The authors verified each claim against the data and take full responsibility for the content, including any error that remains.

REFERENCES

Rui Abreu, Peter Zoeteweij, and Arjan Van Gemund. An evaluation of similarity coefficients for software fault localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, 2006. doi: 10.1109/prdc.2006.18.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Hanning Chen, Keyu Man, Kevin Zhu, Chenguang Zhu, Haonan Li, Tongbo Luo, Xizhou Feng, Wei Sun, Sreen Tallam, Mohsen Imani, and Partha Kanuparthy. Torchtraceap: A new benchmark dataset for detecting performance anti-patterns in computer vision models. *arXiv preprint arXiv:2512.14141*, 2025a.

Xiaoyi Chen, Siyuan Tang, Rui Zhu, Shijun Yan, Lei Jin, Zihao Wang, Liya Su, Zhikun Zhang, XiaoFeng Wang, and Haixu Tang. The janus interface: How fine-tuning in large language models

amplifies the privacy risks. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1285–1299, 2024. doi: 10.1145/3658644.3690325.

Zhaoling Chen, Xiangru Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, and Arman Cohan. Locagent: Graph-guided llm agents for code localization. *arXiv preprint arXiv:2503.09089*, 2025b.

Zhi-Yuan Chen, Hao Wang, Xinyu Zhang, Enrui Hu, and Yankai Lin. Beyond the surface: Measuring self-preference in LLM judgments. *arXiv preprint arXiv:2506.02592*, 2025c.

Daniel Alencar da Costa, Shane McIntosh, Weiyi Shang, Uirá Kulesza, Roberta Coelho, and Ahmed E. Hassan. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 43(7):641–657, 2017. doi: 10.1109/TSE.2016.2616306.

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.

Chris Egersdoerfer, Arnav Sareen, Jean Luca Bez, Suren Byna, Dongkuan Xu, and Dong Dai. Ioagent: Democratizing trustworthy hpc i/o performance diagnosis capability via llms. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2025. doi: 10.1109/IPDPS64566.2025.00036.

Yile Gu, Zhen Zhang, Shaowei Zhu, Xinwei Fu, Jun Wu, Yida Wang, and Baris Kasikci. Ekka: Automated diagnosis of silent errors in llm inference. *arXiv preprint arXiv:2606.04594*, 2026.

Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Yin Zheng, Zejian Yuan, and Zejun Ma. Swe-perf: Can language models optimize code performance on real-world repositories? *arXiv preprint arXiv:2507.12415*, 2025.

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 (ICLR)*, 2024.

Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and detecting real-world performance bugs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2012. doi: 10.1145/2254064.2254075.

James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 2005. doi: 10.1145/1101908.1101949.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014. doi: 10.1145/2610384.2628055.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Misoo Kim, Youngkyoung Kim, and Eunseok Lee. Denchmark: A bug benchmark of deep learning-related software. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021. doi: 10.1109/msr52588.2021.00070.

Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. Practitioners' expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016. doi: 10.1145/2931037.2931051.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613165.

J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 1977. doi: 10.2307/2529310.

Hongwei Li and Yongjun Wang. Reliability of llm inference engines from a static perspective: Root cause analysis and repair suggestion via natural language reports. *Big Data and Cognitive Computing*, 2026. doi: 10.3390/bdcc10020060.

Jiaxin Li, Yiming Zhang, Shan Lu, Haryadi S. Gunawi, Xiaohui Gu, Feng Huang, and Dongsheng Li. Performance bug analysis and detection for distributed storage and computing systems. *ACM Transactions on Storage*, 2023. doi: 10.1145/3580281.

Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. Swe-fficiency: Can language models optimize real-world repositories on real workloads? *arXiv preprint arXiv:2511.06090*, 2025.

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.

Ayush Nangia, Shikhar Mishra, Aman Gokrani, and Paras Chopra. Iso-bench: Can coding agents optimize real-world inference workloads? *arXiv preprint arXiv:2602.19594*, 2026.

Adrian Nistor, Tian Jiang, and Lin Tan. Discovering, reporting, and fixing performance bugs. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013. doi: 10.1109/msr.2013.6624035.

NVIDIA. TransformerEngine: a library for accelerating transformer models on NVIDIA gpus, 2025. Software, `https://github.com/NVIDIA/TransformerEngine`.

Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. Fine-grained commit-level vulnerability type prediction by cwe tree structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023. doi: 10.1109/icse48619.2023.00088.

Chris Parnin and Alessandro Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, 2011. doi: 10.1145/2001420.2001445.

José Pombal, Ricardo Rei, and André F. T. Martins. Self-preference bias in rubric-based evaluation of large language models. *arXiv preprint arXiv:2604.06996*, 2026.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020. doi: 10.1109/sc41405.2020.00024.

Giovanni Rosa, Luca Pascarella, Simone Scalabrino, Rosalia Tufano, Gabriele Bavota, Michele Lanza, and Rocco Oliveto. Evaluating SZZ implementations through a developer-informed oracle. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 436–447, 2021. doi: 10.1109/ICSE43902.2021.00049.

Marija Selakovic and Michael Pradel. Performance issues and optimizations in javascript. In *Proceedings of the 38th International Conference on Software Engineering*, 2016. doi: 10.1145/2884781.2884829.

Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. Gso: Challenging software optimization tasks for evaluating swe-agents. In *Advances in Neural Information Processing Systems 38*, 2025. doi: 10.52202/085713-5054.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Suryansh Singh Sijwali, Angela Marie Colom, Anbi Guo, and Suman Saha. Fixing performance bugs through llm explanations. In *2025 IEEE International Conference on Artificial Intelligence Testing (AITest)*, 2025. doi: 10.1109/aitest66680.2025.00020.

Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories (MSR)*, pp. 1–5, 2005. doi: 10.1145/1083142.1083147.

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.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in LLM-as-a-judge. *arXiv preprint arXiv:2410.21819*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. *arXiv preprint arXiv:2212.09561*, 2022.

W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 2016. doi: 10.1109/tse.2016.2521368.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

Haowen Yang, Yun Peng, and Zishuo Ding. Effiholmes: Differential profiling-guided repository level time inefficiency fix localization. *arXiv preprint arXiv:2608.03558*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

S. Zaman, B. Adams, and A. E. Hassan. A qualitative study on performance bugs. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012. doi: 10.1109/msr.2012.6224281.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Reproduction checklist

Released artifacts. The 430 items as (repository, inducing SHA, fixing SHA, label, label source, split) tuples; the 32-label taxonomy with definitions; the 2,941 decoy commits as the 3,642 (decoy, item) rows of the three released strata; the harness that renders items and enforces the leak guard; the scorer; and the append-only run database from which every number in this paper is generated. The release also names the 11 test items whose answer key an author read during the label audit (`results/split_profile.json`, Appendix I), so a test-split report can leave them out. We do not redistribute repository contents, with two exceptions we measured rather than assumed.

First, 15 corpus pair records use a whole sentence of justification as a dictionary key inside `attribution.votes`, and those sentences quote short upstream fragments, so a small amount of verbatim upstream text does ship (`code/release_docs.py` names which pairs).

Second, Figure 2 prints one item's two changed lines verbatim, so that figure ships two lines of upstream source; the repository they come from is Apache-2.0 (`results/licences.json`).

Otherwise items name upstream commits, and the release ships a generated `clone_upstream.sh` that clones them at their pinned commits into the directory the harness reads. Nothing downstream of the saved per-item outputs needs a clone; the release README lists the scripts that do, the committed output each of them hands back, and the exit status each uses to report that the block was not recomputed. More than one status is in use, and none of them means the block was recomputed.

The chain in `code/rebuild_all.sh` is also not one command a reader can run to the PDF inside the archive. The released copy of the corpus keeps only an allowlist of pair fields and `prior` is not one of them, so `code/symptom_prior.py` cannot measure how many symptom lines came from an imported prior; it says so and exits nonzero, and because the chain runs under `set -e` it stops there, in the derived-analysis step, with the table, figure and page-gate steps never reached.

The `MANIFEST.json` that packaging writes beside the archive carries the exit code, the step reached and the tables that ship as generated rather than rewritten, among them the independent-relabelling table (Table 41) and the model-id table (Table 3), all measured when the archive is built; the archive's own `README.md` lists the reused outputs and points at that manifest.

`code/retry_audit.py` is the one fallback that withholds numbers instead of reusing them: with no call cache and no committed output to reuse it omits the per-call counts rather than writing zeros, so the macros that read them print `??` instead of a measurement.

Two further release documents are generated from the harness and the corpus rather than described in prose. `results/answer_v1.schema.json` is the field-level schema of the diagnosis submission, extracted from the harness source so it cannot drift from what the models were shown (Appendix C). `results/release_docs.json` carries one structured attribution-evidence row for each of the 622 pairs, 430 of which are in a split: which channels voted, which abstained, which wrote a verdict, and whether line blame was consulted (98% of pairs, counting a pair only where the blame channel did more than abstain).

The corpus's free-text attribution notes are *not* released. They run to a median of 2,296 characters per pair and quote upstream source verbatim, 7,119 quoted fragments in all, which are excluded under the release policy described above; the structured rows carry the procedure without the quotations.

One piece of that provenance is unrecoverable rather than withheld: the attribution pass recorded its channel votes and its prompt version but not its annotator model, on all 622 pairs, so unlike the mechanism labels (Appendix I) there is no field to read and we cannot rule out that the author-inferred links were inferred by one of the evaluated models.

Environment. Python 3.12. Models are reached through one API gateway; every request is cached on disk under a key that is the SHA-256 of (model id, full request body, cache salt), so a rerun with an unchanged configuration is free and returns byte-identical text. The request body enters the digest whole: all 4 of its top-level fields, including the 4 generation fields and the rendered item, and `maxTokens` is one of them, so rerunning the same prompt at a different output cap is a fresh paid draw rather than a cache hit. One case escapes that, and the client creates it: the cap is clamped to 8,192 for the one model with a lower platform limit *before* the digest is taken, so for that model any two nominal caps above the limit hash to the same key and the second is a cache hit. The cache shows it directly: 320 of that model's cached requests stop at 8,192 and 0 of them carry the 32,000 the run database records for it. The first 40 hexadecimal characters of the digest are the file name. `code/release_docs.py` extracts all of this from the client's source and from the cached request bodies rather than restating it. This is the only determinism available: the API itself is not bitwise reproducible even at temperature 0, which is why we never pool two runs of the same configuration and always report the item-level pairing.

Generation configuration. Temperature 0 where the model accepts it; the models that reject a temperature field are called without one. That fallback is not recorded: no run row in the released database carries a temperature field at all, so which of the two request bodies a given cell used has to be inferred from the model identity rather than read off the row, and since the response cache is keyed on the full request body, a reproducer cannot reconstruct our cache key. The mapping needed to make that inference is released as `results/model_ids.json`: for each short name it

gives the provider model id and whether that model accepts a temperature field; Table 3 prints it. 2 of the 9 ids the gateway serves pin a dated snapshot and 7 are undated aliases, so which weights a provider served behind an alias is outside our control and outside the bundle. The mapping is also wrong about the fallback for 1 of them: it records that model as accepting a temperature field, and all 3,061 of its cached requests were sent without one, because the client drops the field after the provider rejects it once and keeps that only for the life of the process. So 4 models were called with no temperature field where the released mapping marks 3, and the request body cannot be inferred from the mapping either; the per-model request counts in `results/release_docs.json` are the only record of what was sent. Nor is there a call timestamp anywhere in the run database, so the most we can say is that all 136 live runs were called no earlier than 2026-08-22 and no later than 2026-09-18, with 0 of them unbounded on either side. That window is reconstructed, not logged: its lower end is the date the corpus pinned the commits under test, before which the items did not exist, and its upper end is, per run, the earlier of the modification time of its per-item output file and the author date of the first commit that put the run in the database (`code/call_provenance.py`, `results/call_provenance.json`). Output cap 32,000 tokens. The HTTP timeout scales with the cap (90 seconds plus 12 seconds per thousand tokens). At a fixed 180 seconds, Opus 5 at the raised cap exceeded the timeout on most calls and burned six retries per item without ever failing loudly.

Where the 622 candidates came from is documented but not reconstructible from this archive. The funnel below starts at 622 and every row of it can be re-derived from what we ship. The step *before* it cannot. Those candidates were mined from the four repositories' full histories by a pass whose inputs live under `import/` in the corpus repository, which the archive does not carry, and whose predicates are specified in the corpus contract `docs/SCHEMA.md`, which is not published here because it names individuals. A reader who wants to redo the mining has `code/census.py`'s own keyword predicate and the four pins, and will land on a different candidate set of their own: we do not claim otherwise. The consequence to keep in mind is that the largest single loss in the funnel, the 197 cards the label pass left unplaced, is measured *within* a candidate pool whose own selection bias we cannot bound here. Mechanical filters downstream of that pool are reported per row below; selection upstream of it is not.

Table 2: From 622 candidate commit pairs to 430 items. All but one of these filters are mechanical and run before any model does. The exception is the row that asks whether the corpus supplies a mechanism label: a card carries one only where Claude Opus 5's deep-read pass placed that commit under a mechanism, and that pass left 197 of the 622 cards unplaced, the largest loss in the funnel. The model that decided it is the one that ranks first in Table 1. Tier A is 219 of the 622 candidate pairs, tier B 338, and the excluded tier C 65. Those counts are before the eligibility filters; 164 tier A and 266 tier B pairs survive into the 430 items, which is the split the main text reports. The diff-size filter caps the inducing diff at 2,000 lines, but the prompt shows only the first 900 of them (§2), so the two numbers are not the same population: 108 of the 430 released items and 82 of the 344 scored ones, 24%, were answered from a clipped diff. Fix-site accuracy on those items runs 0.146 to 0.296 across the 6 models against 0.351 to 0.659 on the rest. That gap does not separate a harness artifact from item difficulty, because a diff past 900 lines is a harder localisation target whatever the harness shows and the two subsets are different items, so we print no test for it. What it does establish is that the fix-site column is not a reading of localisation alone on that quarter of the pool. `code/diff_trunc.py` writes `results/diff_trunc.json`.

Filter	remaining	lost
candidate (inducing, fixing) pairs mined from the four repositories	622	–
both commits present in the clone	622	0
corpus supplies a mechanism label	425	197
attribution tier A (fix names or reverts the inducing commit) or B (inferred)	395	30
inducing commit is not a merge	395	0
fix and inducing commit share at least one product-code file	372	23
inducing diff at most 2,000 lines	335	37
fix has at least one hunk with an extractable symbol	334	1
+ items relabelled by the cross-vendor panel	96	–
PerfDiag	**430**	–

Table 3: Every model id the released mapping `results/model_ids.json` carries, 15 rows, and what can be pinned about each. The rows that are not models are the zero-call baselines: they name no provider id and print a dash in every column. Of the rest, the 9 rows the API gateway serves give the id in the exact form the API takes, which is what a reproducer needs to rebuild a request body, and the 4 rows that run locally give a weights repository id instead. The short name in parentheses is the string the run database stores, so it is the key to join on; the printed name is what the tables use, and a row the paper does not name individually prints only the short name. Not every row is a model we evaluate: besides the zero-call baselines, `sonnet46`, `opus48` and `mistrall` appear only as the 3 adjudicators of the blind link audit (Appendix F), and they are listed here because that audit produces numbers in this paper and re-running it needs their ids pinned too. An earlier version of this table omitted them, which left the one place a reader can check model versions silently incomplete for the one experiment whose models are not in Table 1. *Takes temp.* is measured from the cached request bodies rather than read off the mapping: 4 models were sent no `temperature` field at all, and † marks the 1 row where the mapping records that the model accepts one although every one of its 3,061 cached requests carries none, because the client learns that at runtime and writes it nowhere. *Snapshot* is the caveat this table exists for: only 2 ids contain a date, 4 are pinned by a revision and a weight digest, and 7 are undated aliases whose weights we cannot pin, then or now; the baseline rows carry no snapshot of any kind. Rows are in alphabetical order of the short name; this is a lookup table and the order is not a ranking. `code/release_docs.py` writes it from `results/model_ids.json`, which `code/model_ids.py` writes from the client's own constants, and from the response cache.

Model (short name)	Provider model id	Takes temp.	Snapshot
char-*n*-gram classifier (`bow-char-lr`)	–	–	–
`emb-unixcoder-mean`	`microsoft/unixcoder-base`	–	by revision
`emb-unixcoder-meanmax`	`microsoft/unixcoder-base`	–	by revision
`emb-unixcoder-site-top1`	`microsoft/unixcoder-base`	–	by revision
`emb-unixcoder-site-top3`	`microsoft/unixcoder-base`	–	by revision
GPT-5.6 (`gpt56`)	`us.openai.gpt-5.6-terra`	no†	alias, no
Haiku 4.5 (`haiku45`)	`us.anthropic.claude-haiku-4-5-20251001-v1:0`	yes	by date
Llama-4-Maverick (`llama4mav`)	`us.meta.llama4-maverick-17b-instruct-v1:0`	yes	alias, no
`mistrall`	`us.mistral.pixtral-large-2502-v1:0`	yes	alias, no
`opus48`	`us.anthropic.claude-opus-4-8`	no	alias, no
Opus 5 (`opus5`)	`us.anthropic.claude-opus-5`	no	alias, no
`pathfeat-lr`	–	–	–
Sonnet 4.5 (`sonnet45`)	`us.anthropic.claude-sonnet-4-5-20250929-v1:0`	yes	by date
`sonnet46`	`us.anthropic.claude-sonnet-4-6`	yes	alias, no
Sonnet 5 (`sonnet5`)	`us.anthropic.claude-sonnet-5`	no	alias, no

Population filters. Two rows of Table 2 move the population rather than clean it. The first is the mechanism-label row: 425 of the 622 cards carry a label and 197 do not, and what separates them is a judgement by Claude Opus 5. A commit whose mechanism that model could not place is a commit the corpus-labelled pool does not contain, so that pool is easier for that model, and for models that answer like it, than the candidate population is, and the 334 corpus-labelled items inherit the effect. The cross-vendor panel reverses part of it and not all: 129 of the 197 unplaced cards went to the panel and 96 of those got a majority label and entered the benchmark, so the commits no annotator could place are still absent, and `code/label_authorship.py` fails if the panel pool ever stops being a subset of the unplaced cards. The second is the diff-size cap, the largest loss among the eligibility rows: it removes the widest inducing commits, so a fix-site number measured here speaks

to localised regressions and not to the bulk rewrites the cap excludes. The attribution row and the shared-file row push in the same direction, towards regressions whose fix touches a file the inducing commit touched, and both are smaller.

Eligibility on both annotation branches. Every row of the funnel is a pure function of fields the frozen census table already carried, so the panel-labelled items can be put through it. An earlier version of this appendix said they could not be, and that sentence was wrong; we withdraw it. `code/panel_eligibility.py` copies the predicates from `code/census.py`, reproduces the corpus column of Table 2 cell by cell, and only then reports the panel branch (Table 4). Two facts could have come out otherwise. The panel candidate pool is exactly the unplaced cards that pass all 6 mechanical filters, asserted as a set identity against the released pool file (129 items, symmetric difference zero); and the eligibility fields of the 96 panel items are identical between the census frozen at `9c0c014e` (2026-09-06, an ancestor of `35310398`, the commit that added the panel votes) and the census written after the panel ran, 0 mismatches. Given those two the headline is a consequence rather than a second measurement: 96 of the 96 released panel items pass all six mechanical filters with 0 filter failures, and the only row they fail is the branch row itself, which they fail by construction because the pool was selected by the *absence* of a card label.

Differences between the eligible populations. Four numbers run the other way and the funnel table should not be read without them. First, 3 of the 5 mechanical filters that follow the branch row remove a larger fraction of the panel branch than of the corpus branch: the attribution row removes 18% of the panel branch against 7% of the corpus branch, and the shared-product-code-file row and the diff-size cap go the same way, the two exceptions being the merge row and the extractable-symbol row, which remove almost nothing on either branch. Together those filters leave 65% of the panel pool against 79% of the corpus pool: the released panel items clear the same filters, but they are a more heavily selected slice of a pool that was mechanically worse to start with. Second, the test split is 49% tier A (42 of 86) against 35% (122 of 344) on the pool every number in this paper is measured on, and tier A is the easier stratum for the strongest model (0.631 against 0.559 on families, 0.697 against 0.500 on fix sites), so a test score is optimistic on a third shift axis that Table 9 does not carry. Third, only 20 of the 344 scored items are panel-labelled against 76 of 86 in test, so this check lands mainly on the split nobody has scored, and the panel branch is 22% of the benchmark. Within the released items the panel branch is also the more tier A of the two, 51 of 96 (53%) against 34% of the corpus-labelled ones. Fourth, the panel branch sits closer to the diff-size cap in the tail: its p90 inducing diff is 1,181 lines against 954 on the corpus branch, while the medians go the other way (143 against 186), so the two branches differ in the shape of that distribution and not only in its centre. A user who needs the eligibility criteria to hold on every item can read the panel columns of Table 4; a user who needs the two populations to be alike should still read the corpus column of Table 11 and not the pooled one.

Leak-guard implementation. Every read the model can perform goes through a repository view pinned to one revision: the *parent* of the inducing commit. Three rules are enforced at the point where a revision string is resolved, before any git command runs.

First, the model is never given a revision to pass. The tools take a path and, for history, a count; there is no argument through which a SHA, tag, branch or `HEAD` can be supplied. This is stronger than filtering revisions, because there is nothing to filter.

Second, the pinned revision is checked at view construction: resolving the inducing commit, the fix commit, any descendant of either, or `HEAD` raises `LeakGuard`. The check is on ancestry, not on string equality, so a different spelling of the same commit does not slip through.

Third, a raised `LeakGuard` voids the whole run rather than the single call. A run that touched forbidden state is not partially usable, and making it fail loudly is the only way the count we report (0 attempts in 1,605 repository tool calls) means anything. That count covers the repository access there was to guard. Of the 187 recorded runs that ship a per-item file, 4 ever built a repository view, the two evidence-tool variants of Appendix G, over 268 item-runs; the other 183 are single prompt calls with no repository view at all, so the guard had nothing to fire on. Read the 0 as covering the 1,605 calls that did pass through it, not as 187 independent opportunities.

Table 4: **The eligibility funnel on both branches.** The corpus columns reproduce Table 2 cell by cell; the panel columns invert the mechanism-label row, which is what defines that branch, and leave the other 6 filters unchanged. The last column re-checks every released panel item against each filter independently, so it is not the sequential loss: its only non-zero cell is the branch row. † the row that defines the panel branch rather than a quality criterion: the panel pool was selected by the *absence* of a card label, so all 96 released panel items fail this row by construction, which is what the last column prints. The last row before the total counts items that reached a panel majority, not cross-vendor majorities: the panel is 3 models from 2 vendors, 89 of the 96 majorities span vendors and the other 7 were formed by two models from one vendor (Appendix I). Predicates are copied from `code/census.py`; `code/panel_eligibility.py` writes `results/panel_eligibility.json` and refuses to emit the table if the corpus column stops reproducing the frozen chain.

	corpus branch		panel branch		released
Filter	remaining	lost	remaining	lost	panel fails
candidate pairs	622	–	622	–	0
both commits present	622	0	622	0	0
corpus supplies a mechanism label†	425	197	197	425	96
attribution tier A or B	395	30	162	35	0
inducing commit not a merge	395	0	162	0	0
share a product-code file	372	23	147	15	0
inducing diff at most 2,000 lines	335	37	129	18	0
fix hunk with an extractable symbol	334	1	129	0	0
three-model panel majority	–	–	96	33	–
released items	**334**	–	**96**	–	**0**

The harness ships a self-test that asserts all four cases: the parent resolves, and the inducing commit, the fix commit and `HEAD` each raise. We ran it before every sweep.

Reproduction pipeline. Everything downstream of the model calls is one script, and it fails as a whole if any step fails. Each check below was added after a version of the paper shipped a number that the check would have caught. Step 5's test-split gate is a gate on *rows*: it is what makes "no model output on a test item" true, and it says nothing about the 11 test answer keys the label audit read (Appendix I).

```
# the model calls, the steps that cost money, all
# from ideas/I0021-perfdiag-decision-graph/.
# six-model main table, M3 ladder, closed book:
bash probe/tonight5.sh
# the injection sweep behind Figure 1(b):
bash probe/breakeven32.sh
# the three decoy strata on three models:
bash probe/decoys3.sh
# the commit-message arm at the matched cap:
bash probe/msg32k.sh
# the two overnight variant searches (S ids, then
# B ids), and the two arms added afterwards:
python3 probe/search.py
python3 probe/search2.py
bash probe/extra_arms.sh
# the retired 4,096-cap pass, kept because the
# database keeps its rows:
bash probe/run_plan.sh
bash probe/run_plan2.sh
# everything else in one shot, from
# projects/I0021-perfdiag-decision-graph/:
bash code/rebuild_all.sh
```

```
#   1 two-pass label agreement (CPU only)
#   2 run files -> runs.jsonl (append-only)
#   3 retire runs covered by a larger one, by a
#     binding output cap, or by a re-scoring
#   4 re-run the zero-call runs in a clean-tree
#     copy, compare byte-for-byte, then re-stamp
#     provenance for rows whose code was not yet
#     committed when they were written
#   5 validate: 0 test-split rows, 0 rows whose
#     code revision was dirty
#   6 tables + number macros, and every macro the
#     text cites must exist
#   7 figures
#   8 typography: no table wider than its column,
#     no overfull box, page count exact
```

Cost per evaluation. 344 items, one call each, no execution and no GPU. Input is 3.35 to 5.43 million tokens depending on how much of the diff the model's tokenizer charges for, and output is at most 2.71 million: the strongest models write the most, because they explain the mechanism at length before naming it. This is the practical argument for the execution-free design as much as the hardware one: a benchmark that needs an H100 node per item cannot be re-run on every new model release, and this one can.

Cost. 24,018 model calls, 281.14 million input tokens and 50.68 million output tokens across every run in the database, including voided and superseded ones. Table 5 breaks this down by model. There is no GPU cost: nothing is executed, and all 9 zero-call arms are CPU-only: the pretrained-encoder arms cost at most 7 CPU-hours on 32 CPU threads (Appendix G.5), and CPU-only there is a choice rather than a hardware limit. The sum is taken over distinct run files, not over database rows: re-scoring a run under a corrected metric definition and re-stamping one under a clean code revision each append a row that copies the original run's token usage, so summing rows would count the same tokens more than once. 440 rows collapse onto an earlier row this way.

Table 5: Cost by model over every distinct run file, including runs later voided or superseded. Input-token counts are as reported by the API except where marked †: GPT-5.6's Converse responses return a single-digit `inputTokens` regardless of request size, so for that model we substitute a per-call estimate of request bytes over four, giving 24.90 million input tokens instead of the reported near-zero. DeepSeek-R1's row is zero on purpose: all of its calls were rejected at the API for lack of tool support, so nothing was billed.

Model	item-runs	model calls	input MTok	output MTok
Claude Sonnet 4.5	8,710	13,057	148.26	15.30
Claude Opus 5	3,856	4,949	65.15	24.28
GPT-5.6	2,174	2,428	24.90†	3.91
Claude Sonnet 5	1,005	1,378	21.20	5.65
Claude Haiku 4.5	902	1,036	12.08	1.29
Llama-4-Maverick	902	1,036	9.55	0.25
DeepSeek-R1	134	134	0.00	0.00
bow-char-lr	1,376	0	0.00	0.00
top1hunks	1,376	0	0.00	0.00
top3hunks	1,376	0	0.00	0.00
majority	1,376	0	0.00	0.00
total	–	24,018	281.14	50.68

B THE LABEL SET

Table 6 lists all 32 mechanisms in 11 families and the first sentence of each definition. Full node descriptions average 815 words across all text fields. They include exclusion criteria, static and runtime evidence, confirm/refute predictions and observed fix patterns. The release stores them in `graph/graph.json`, under `nodes[].definition` and the adjacent `evidence`, `verification` and `fix_patterns` fields. These definitions specify the distinctions used by the scorer; the table provides an overview rather than a complete account of neighbouring labels.

Table 6: All 32 mechanisms grouped by cause family, with the first sentence of each released definition. Definitions are truncated at 300 characters here; the full record of a node averages 815 words and ships with the benchmark, in `graph/graph.json`.

Mechanism	One-sentence definition (first sentence of the released definition)
Misplaced ordering between streams `async-overlap-ordering-wrong`	
Two things that progress asynchronously	
`hook-and-callback-order-vs-state`	The diff changes when a host-side Python callable fires relative to the state it observes – the site where a hook is registered or invoked, the argument that narrows which pending entries a callback drains, the placement of a per-iteration counter reset the callable depends on, or whether the . . .
`missing-cross-stream-ordering-edge`	At the site where asynchronous work is enqueued, the edge that orders a producer against its consumer is wrong: the kernel or collective is issued on the default stream rather than the caller's current stream, or the event record / cudaStreamWaitEvent / wait_stream / record_stream / async-handle . . .
`stream-event-boundary-missing`	Every collective in the hunk is left exactly as it was and only the boundary of the communication region around it is wrong: its exit is never closed, because the forked communication stream is only ever a target of stream-wait-event and has no event recorded on that stream itself for the main . . .
Wrong size or budget `capacity-accounting-wrong-quantity`	
An expression that produces a capacity quantity	
`admission-budget-over-wrong-population`	A running or aggregate quantity that a scheduler, allocator or placement path consults before it acts – a per-group admission count, a bucket-occupancy or block-reference tally, a remaining-token or scan bound, a per-stage/per-engine share table – is accumulated over the wrong set of items . . .
`placeholder-shape-sizes-the-budget`	A one-shot sizing expression that feeds an allocation or a capacity reservation – a pad width, a dummy-profiling tensor extent, a per-page byte size, a KV-cache byte budget – is evaluated from the wrong input: a configured or captured maximum, a cache-wide or batch-wide extent, or a profiling . . .
Collective over the wrong group `collective-scope-or-placement-wrong`	
A collective, or a resource that must be private to one worker process, is issued or bound over the wrong set of participants	
`collective-group-membership-wrong`	At the point a transfer is issued, the participants it reaches are wrong: the group or destination-rank argument handed to the collective names a broader default group or a different communicator than the ranks that own the reduced quantity, or the communicator object behind it is built from a . . .

Table 6, continued

Mechanism	One-sentence definition (first sentence of the released definition)
`per-worker-resource-binding-wrong`	The value that decides which physical resource one worker process claims for itself – the cuda index it hands to set_device or to init_process_group's device_id, the visible-device mask a launcher exports around a child engine's start, the NUMA/CPU list it hands to thread affinity, or the . . .
Work escapes the compiled region `compiled-region-does-not-cover-the-work`	
A compiled or capture-based execution path for these operations already exists in the tree, but at parent_sha the work escapes it	
`captured-artifact-specialized-for-what-the-runtime-never-asks`	Every compile/capture call site and bypass branch is left where it was; instead the diff corrects what the artifact is built for or when it may be reused – the enumerated warm-up/capture shape or size list (or the buffer and max-length quantities the served shape is derived from), the flag, set or . . .
`region-boundary-excludes-executed-work`	The diff changes which code the compiler or the capture stream ever sees – deleting a graph break, a 'do_not_compile'/'enforce_eager' bypass or an eager-fallback branch, making a 'torch.compile'/'support_torch_compile' site live where it was absent or commented out, registering a . . .
Setting never takes effect `expressed-intent-not-reached`	
Both the behaviour the author expressed and the behaviour actually executed already exist and are reachable in the tree, but the wiring that is supposed to select between them	
`callee-binding-misresolved`	At a call boundary the caller's resolution lands on something other than the implementation that already exists in the tree – the callee declares its name, key or capability under one spelling while the reader consults another, the interception or wrapper meant to redirect construction/forward was . . .
`consumer-reads-unconnected-config-site`	At the consumer, the field, key, parameter or flag the machine actually reads is not the one the caller's value was written to – the read names a different attribute, a lookup key present in no registered entry or shipped file, a locally recomputed substitute that the signature never receives, or . . .
`default-clobbers-expressed-value`	The write site and the read site name the same configuration field, but the value the machine finally acts on is decided by the order and scope of resolution rather than by the caller: a default, auto-derived, profiled or per-module assignment on that field executes after the caller's value is . . .
`gate-reads-wrong-operand`	An enable-or-skip predicate that already exists at parent_sha is applied to an operand or grouping that cannot carry the configured choice – an always-present configuration container instead of the field inside it, a raw integer or per-rank list where a comparison or an aggregate belongs, an . . .
`stage-skip-keyed-on-wrong-signal`	A well-formed predicate that decides whether an optional stage runs, or which of two implementations a call dispatches to, is keyed on a runtime signal that only correlates with the condition the branch body actually depends on – a model-, arch-, version- or parallelism-level property standing in . . .
`unconditional-fallback-pins-slow-path`	At a site that selects between a slower conservative implementation – a fallback, a deterministic or reference variant, a workaround shim, or a reject branch that routes the run onto an older engine – and a faster implementation already present in the tree, the selection carries no predicate the . . .

Table 6, continued

Mechanism	One-sentence definition (first sentence of the released definition)
Wrong answer about the hardware `hardware-capability-misjudged`	
The expression that decides what the machine or the build itself can do	
`build-time-arch-specialization-gap`	A value that is frozen when the binary or extension is produced – the arch list a source is gencoded for, a preprocessor guard on an architecture or toolkit-version macro, a per-architecture occupancy constant feeding '__launch_bounds__', the compile-flag list handed to an extension, or the . . .
`device-capability-probe-mispredicated`	A boolean the process computes at run time from values it reads out of the machine or the build – a compute-capability pair, a device name or platform/vendor identity, a CPU feature list, a toolkit or library version string, the importability of a module or symbol, a class's declared support flag . . .
`kernel-availability-probe-misresolves`	At the site that chooses a specific named fused or vendor kernel over its reference implementation, the expression deciding whether that kernel is usable here – the symbol the availability probe references, the module path it imports, the polarity of the presence test, the version bound on the . . .
Operand form wrong for the kernel `operand-form-mismatched-to-kernel`	
A kernel or operator is handed something in a form it cannot consume as-is: either an operand whose physical representation	
`launch-geometry-from-wrong-default`	The diff changes how a scalar launch-configuration value is resolved – a default-config table entry, a dispatch threshold or floor, a derived heuristic, or a checked-in autotune result – for tile shape, warp/stage count, KV-split count or KV page/block size, so that a given shape and device is . . .
`operand-form-normalized-before-kernel`	The diff changes the physical form in which a tensor operand reaches its consumer – its layout, stride, contiguity or varlen packing, or its element type and quantized form – either by making the producer emit the form the consumer reads or by adding or deleting a whole-tensor transform between . . .
Ranks disagree at a rendezvous `ranks-disagree-on-the-rendezvous`	
Each side of one rendezvous derives its own half of it from state visible only to itself	
`collective-emission-guard-wrong`	A distributed call site and its argument list are left textually unchanged and only the statement-level gate enclosing it is wrong – the branch condition or indentation level it sits under, an early return or per-rank hook that lets some ranks skip it, the mutual exclusion around the region that . . .
`cross-rank-rendezvous-condition-divergent`	The diff changes a quantity the synchronization point itself consumes so that every participant derives it the same way – the element count or dtype of the buffer posted to a collective, the process group or rank grid the collective runs on, the operand a rank must supply to enter the collective . . .
Stale decision replayed `recorded-decision-diverges-from-runtime`	
A value the tree records once	
`decision-resolved-once-then-replayed`	A control-flow or dispatch value – a predicate, a backend/kernel/algorithm or launch-config selection, a bound callable, or the ordering of callables baked into a graph capture – is resolved once at construction, first call, or capture time and stored under a key that omits state it depends on . . .

Table 6, continued

Mechanism	One-sentence definition (first sentence of the released definition)
`warmup-bound-buffer-reused-against-changed-state`	A memory object – a preallocated device buffer, a static capture input/output slot, a recycling-pool entry, a warm-up dummy activation or the tensor list handed to a traced region – is allocated and bound once in a preparatory phase (warm-up, profiling, graph capture, pool hand-out) or carried . . .
Resource outlives its phase `resource-lifetime-outlives-its-phase`	
Storage or an OS handle stays reachable past the phase or unit of work that needed it	
`allocator-pool-not-reset-at-phase-boundary`	Storage stays checked out of the caching allocator across the boundary of the phase that was supposed to bound it – a graph-capture region whose frozen collector is released without any collection, a memory-pool or sleep/wake context opened around work that allocates nothing, a process-global . . .
`persistent-buffer-binding-or-extent-wrong`	A long-lived buffer that a producer and a consumer are meant to share is reserved from the wrong extent, laid out at an offset derived from a per-call quantity, bound permanently to the first call's shapes or to the wrong element of a collection, or bypassed entirely by a per-step allocation of . . .
`retire-path-release-missing`	A process-lifetime-scoped owner – a dict/set/list of per-request or per-layer state, a self-keyed memoization cache, a global hook registry, an attribute left on a surviving parameter, a socket or a child-process handle – keeps a reference to one unit of work or one object after that unit is . . .
Avoidable work on the hot path `steady-state-path-carries-avoidable-work`	
On a path the parent tree already reaches once per training step, per transformer layer, per request or per op invocation, the code executes work whose outcome the running configuration does not need	
`blocking-handoff-between-concurrent-executors`	On the per-step or per-request path, one host execution context waits for another – a 'Future.result()' on a dispatched model execution, a 'queue.get(timeout=...)' taken without first testing emptiness, a 'while cond: time.sleep(...)' loop guarding a shared buffer, or a repeated poll of a . . .
`invariant-setup-hoisted-or-memoized`	A value or object whose content cannot differ between invocations – a version-string parse, a device-property query, a workspace or pinned-buffer allocation, a per-call dispatch/shape-marking wrapper, a deep copy of a parameter or metadata object, a tokenizer or executor pool – is constructed . . .
`per-element-loop-batched-or-folded`	Work that is genuinely required and whose result is consumed is issued at the wrong granularity – one device call, GEMM, state copy or file read per element of a runtime-sized collection (per expert, per cache slot, per request, per dataset item) inside a host loop, or one standalone . . .
`per-step-device-to-host-readback`	Host code on a path that runs unconditionally – a per-step attention-metadata build, a per-module forward or state_dict hook, a per-message serializer, or module-scope import / platform-probe code – reaches across the host-device boundary: it pulls a device-resident value into host control flow . . .

Table 6, continued

Mechanism	One-sentence definition (first sentence of the released definition)
`redundant-work-deleted-or-gated`	A statement on the per-forward, per-step or per-request path computes, copies or zeroes something that the running configuration never consumes – the value feeds only an assertion, a print or a log-line guard; it is an operation that is a no-op at that configuration's parameter value; it refills a . . .

C Full configuration of every arm

Arms. FLAT presents all 32 labels at once and asks for one answer. FLAT_EVIDENCE is the same with read access to the repository at the inducing commit's parent (read file, grep, log; up to six tool turns), which is the interleaved reason-and-act loop of Yao et al. (2022) with the action set restricted to reads. GRAPH walks the three-layer diagnostic graph one layer at a time, so it is a decomposition scaffold in the sense of Zhou et al. (2022) and Khot et al. (2022), with the sub-questions fixed in advance by the taxonomy instead of proposed by the model. GRAPH_EVIDENCE adds repository access to that walk. GRAPH_VERIFY asks for a discriminating prediction at each layer before descending, which is self-verification (Weng et al., 2022) applied once per layer. GRAPH_BEAM keeps the top-b branches at each layer, a beam search over the tree that Yao et al. (2023) searches with a model-generated value estimate. FREEFORM asks for a free-text diagnosis and maps it onto the taxonomy afterwards. SYMPTOM_ONLY asks for the top layer alone. The knobs applied on top of these arms carry their own sources: chain-of-thought (Wei et al., 2022), few-shot demonstrations (Brown et al., 2020), and self-consistency voting over samples (Wang et al., 2022).

Evaluation units and validity criteria. An *arm* is a prompt shape (flat, graph, freeform). A *variant* is an arm with one knob changed. A *cell* is one (variant, model, split) combination. A cell is *void* if under 90% of its answers parse or over 10% fail to finish; void cells appear in every table with their numbers, marked, and are never dropped, because dropping them would make the parse gate a filter on results rather than on validity. One evaluation of PerfDiag is 344 items at one call each, 3.35 to 5.43 million input tokens across the 5 models whose provider reports input tokens, and no GPU.

Call budget and continuation requests. Every cell has the same budget, 2 model calls per item. The second call is not a resample. The harness issues it only when the first response yields nothing the scorer can read, meaning no `submit_diagnosis` tool call and no JSON in the text, and it issues it as a continuation of the same conversation: the first response stays in context and two instructions are appended, call the tool now rather than replying in prose, and this is the last turn. Because the trigger is a property of the response and not a per-model setting, the budget is uniform across models and the spend is not: 19 of 344 items for GPT-5.6, 2 for Opus 5, 1 for Sonnet 5, and none at all for the other 3 models. That is a per-model difference inside the headline table, so we measure it here rather than average it away. `code/retry_audit.py` replays every second call out of the request cache, which stores the request body and the response for each call, and writes the per-item reasons to `results/retry_audit.json`: 17 first responses returned no visible content at all while still billing output tokens, 4 returned text but never called the tool, and 1 hit the output cap. It is not best-of-two, and the evidence for that is negative: across the 22 cases the first response contained a submittable answer 0 times and named a label from the menu 0 times, so there was never a second candidate to prefer, and in all 22 cases the scored answer is the one the second call produced. The effect of the policy is bounded by scoring those items wrong instead, which is what a one-call cap would do given that an unparseable answer counts as wrong: cause family goes from 0.453 to 0.442 for GPT-5.6 and from 0.584 to 0.578 for Opus 5, the other four models do not move at all, the largest change is 0.012 (GPT-5.6) and the ranking is unchanged. The audit also identified two qualifications. Opus 5's 2 items are one distinct prompt: the two commits share a parent and an inducing diff, so one cached response is being counted twice. And Sonnet 5's single continuation is the one call in this table that hit the 32,000-token cap, so that cap is nearly, but not entirely, non-binding.

Uncertainty in break-even thresholds. Both crossings in §5 are read off a curve whose interior points are cache replays of the two endpoints under one per-item hash mask, so the crossing a single sweep reports carries that draw's noise, which is why §5 reads the crossing with that mask averaged out. On the matched 59-item denominators, the flat read-off sits below *the crossing that redraw produces* in 70% of mask redraws for Sonnet 4.5 and 50% for Opus 5, so the Opus 5 side of that line is a coin flip. (The comparison has to be made redraw by redraw: the mask-averaged crossing and the flat read-off are both single numbers, so neither varies under redrawing and no fraction could be read off them.) The split an earlier version of this paper printed, one router clearing Sonnet 4.5's threshold and missing Opus 5's, holds in 36% of paired item resamples with the mask held at the draw the sweep took. The threshold-to-router ratio comparison reverses with the estimator: the single mask we drew gives 1.66× against the router's 1.50×, while averaging the mask out puts the threshold ratio at 1.41×, below the router's. For these reasons, §5 makes no claim about which flat router crosses its threshold, the ordering of shortfalls, or the size and direction of the ratio comparison. `code/breakeven_order.py` writes `results/breakeven_order.json`.

Answer schema and enforcement. A model submits an answer through `submit_diagnosis`; if it returns the object as text, the harness parses that text instead. The schema has 8 fields: `introduces_regression`; `node_id` and `alternative_node_id`, for the primary and second-choice mechanisms; `confidence`; `mechanism`, a two- to four-sentence explanation of the work added or removed and its code path; `sites`; and `confirm_prediction` and `refute_prediction`. Each site contains `file`, `line`, `symbol` and `why`. Only `file` is required within a site, and requested line numbers use pre-fix coordinates. Diagnostic framing requires 6 fields; neutral framing requires 1, allowing a complete answer to stop after `introduces_regression: false`. The released `results/answer_v1.schema.json` is extracted from the harness by `code/release_docs.py`, so a schema mismatch fails the rebuild.

The harness does not enforce either the citation cap or the required-field set. The instruction to cite at most 3 sites appears only in the field description: the schema has no `maxItems`, and the scorer does not truncate the list. Only 4 of 12,316 answers with site lists exceed the cap, but citation budgets vary within it. For the flat calls in Table 1, mean sites per answer range from 1.40 to 2.91, a factor of 2.08. In increasing budget order, Llama-4-Maverick, GPT-5.6, Claude Haiku 4.5, Claude Sonnet 4.5, Claude Sonnet 5 and Claude Opus 5. Llama-4-Maverick cites one site in 232 of 342 well-formed answers; Claude Opus 5 cites three in 314 of 342. That ordering is the raw fix-site ordering of that column apart from two adjacent pairs (GPT-5.6 and Claude Haiku 4.5; Claude Sonnet 4.5 and Claude Sonnet 5). Because any correct citation earns credit, fix-site hit rate is monotone in the citation budget. Section 4.1 therefore compares the weakest model with a one-citation baseline (0.206).

Required-field violations are more common. Of 11,533 diagnostic-framing answers, 770 (7%) fail the released requirements: 38 return `sites` as a JSON string instead of an array, and the rest omit it. None is rejected. The scorer iterates over each string character as though it were a separate site, producing one-character file citations. These answers score zero on every fix-site metric, compared with 68% file-level hits for answers returning arrays.

Parsing alone cannot recover the line-level scores. Among 24 malformed answers in Table 1, 0 pass `json.loads`; the strings contain truncated tool-call fragments rather than serialised arrays. A lenient decoder starting at the first bracket recovers 11, and a filename regular expression finds a path in all 24. However, only 11 contain a `line` key, leaving 13 without recoverable line numbers. We retain the released scoring convention and report an imputed estimate beside the raw score in the table caption.

Optional second-choice predictions. `alternative_node_id` is optional, unenforced and scored: `code/score.py` records whether that second choice equals the gold mechanism as `m1_topk_hit` on every diagnostic-framing item, and no mechanism number in this paper uses it. Models differ on whether they fill it, 0 for Llama-4-Maverick, 76 for Claude Haiku 4.5, 204 for Claude Sonnet 4.5, 287 for Claude Sonnet 5, 241 for GPT-5.6, 308 for Claude Opus 5, out of 311 answers each, so this is the sites cap read in the other direction: an unenforced optional field that would move a scored column rather than one that is open and almost unused. Crediting it turns the exact-mechanism column of Table 1 into 0.177 to 0.653 (Table 35), a gain of 0.164 at its largest, for GPT-5.6. The two events never coincide on an item, because the scorer reads the first choice from `node_id` and the second

from `alternative_node_id`, so the top-2 rate is the sum of the two columns and filling the field cannot move the strict one. **The released leaderboard scores the strict single-leaf match**, and we print the top-2 column so that a later submission reporting it is read as a change of convention and not as progress. A comparison with the two-pass agreement of Table 1 needs a shared denominator, which Appendix I supplies.

Self-reported confidence. `submit_diagnosis` asks for a `confidence` of low, medium or high on every item, and no number in this paper crossed it with correctness until now. Crossed, it is informative for 3 of the 6 models and not for the rest, so it is a property of a model and not of the task (Table 7). The criterion is that among the buckets a model filled with at least 20 items, its most and its least confident bucket have disjoint Wilson intervals in the expected direction; Claude Sonnet 5, GPT-5.6 and Claude Opus 5 pass it. Claude Opus 5 is the sharpest: answering only where it reports high confidence gives 0.789 cause family on 22% of the queue, 0.205 above the 0.584 it reaches on all 344. That is a selective-prediction reading and not a benchmark score, and it is the concrete sense in which the residual is not uniformly hard. The other 3 fail the criterion for two unrelated reasons. Claude Sonnet 4.5 reports high on 330 of 344 items, which leaves no contrast to measure. Llama-4-Maverick leaves the field empty on 313, and on the 25 it does mark high it scores 0.240 against 0.333 on the 6 marked medium, an inversion on counts too small to read as one. Nothing enforces the field: the released harness does not require `confidence` under either framing, which is why one model can leave it empty and nothing rejects the answer. So a PerfDiag number quoted under a coverage constraint has to name the model it came from, because the constraint buys 0.205 on one of the 6 and nothing on 3. We print no test between buckets. They are unpaired strata of one run, the Wilson intervals carry the uncertainty, and a Fisher test here would add a member to the family every other test in this paper is corrected against for a comparison we did not pre-register.

Configuration ranges. View $\in$ {full, no_message, diff_only, closed_book}; label definitions $\in$ {short, full}; scope $\in$ {leaves, families}; starting layer $\in$ {symptom layer (N_0), family layer (N_1)}; extras $\in$ {none, exclusions, fix-patterns, both}; chain-of-thought on/off; few-shot $\in \{0, 3, 6\}$; beam width $\in \{1, 2\}$; symptom oracle rate $\in \{0, 0.5, 1\}$; family oracle rate $\in \{0, 0.25, 0.5, 0.75, 1\}$; self-consistency votes $\in \{1, 3\}$; framing $\in$ {diagnostic, neutral}.

Variant selection and protocol deviation. All search was on dev. One clause was frozen before the first run: keep a variant only if its item-paired McNemar delta against the flat baseline is positive at $p < 0.10$. A second clause, that line-level fix-site accuracy must not drop by more than 5 points, was added when the second overnight search started, so it is not pre-registered and we do not report it as such. Its purpose is that trading localisation for classification is not a win. No variant was rejected by that second clause alone: the one cell whose verdict names it, repository access plus chain-of-thought, also falls below the flat baseline on cause family and its paired test does not clear the threshold either (Table 25), so the frozen clause rejects it as well.

No variant met the pre-registered selection rule. Across the 20 variants for which the paired test is defined the smallest p is 0.0063, but that belongs to `S1-startN1`, which is significantly *worse* than the baseline; among the variants whose point estimate is above the baseline at all (one variant), the smallest p is 0.58. The only significant effect was a decrease in accuracy. We carried one variant forward anyway, supplying the commit message, and we carried it on its point estimate. This follow-up did not meet the pre-registered selection criterion. We nevertheless tested it on 344 items to assess whether its point-estimate gain would replicate: the variant reversed sign on the model it was chosen on and held on a different vendor's model (§4).

Checks on prompt and configuration changes. Four separate times, a knob silently failed to reach the prompt, and each time the affected cell was byte-identical to the baseline while appearing to be a new result, eight of twelve variants in one overnight sweep. Two mechanical guards now run on every cell: the fingerprint recorded for a run is the hash of the prompt *actually sent*, and a variant whose fingerprint equals the baseline's is marked invalid rather than reported; and the generation configuration actually used is recorded and compared against the one requested, with a mismatch raising a warning in the run summary. Both are in the released harness.

Table 7: Cause-family accuracy by the model's own `confidence` field, on the 344 scored items of Table 1, one flat call per item. *not given* is an answer that parsed with the field absent or empty, which the schema permits under both framings. Intervals are Wilson. Buckets below 20 items are printed and excluded from the separation criterion stated above, which 3 of 6 models pass.

Self-reported confidence	n	cause family	95% CI
Llama-4-Maverick			
high	25	0.240	[0.11, 0.43]
medium	6	0.333	[0.10, 0.70]
not given	313	0.323	[0.27, 0.38]
Claude Haiku 4.5			
high	152	0.368	[0.30, 0.45]
medium	63	0.302	[0.20, 0.42]
not given	129	0.318	[0.24, 0.40]
Claude Sonnet 4.5			
high	330	0.370	[0.32, 0.42]
medium	14	0.357	[0.16, 0.61]
Claude Sonnet 5			
high	2	1.000	[0.34, 1.00]
medium	205	0.546	[0.48, 0.61]
low	128	0.336	[0.26, 0.42]
not given	9	0.444	[0.19, 0.73]
GPT-5.6			
high	226	0.513	[0.45, 0.58]
medium	113	0.345	[0.26, 0.44]
low	5	0.200	[0.04, 0.62]
Claude Opus 5			
high	76	0.789	[0.69, 0.87]
medium	234	0.551	[0.49, 0.61]
low	32	0.375	[0.23, 0.55]
not given	2	0.000	[0.00, 0.66]

D Complete run results

The database holds 631 rows, of which 495 are superseded by a later run covering the same items and 136 are live. That is the only partition; voiding is a separate, overlapping label. 61 rows are voided by the parse/finish gate, 49 of them also superseded, and the other 12 still live: a void row is excluded from every aggregate by the gate but nothing has replaced it, so subtracting the voided and the superseded counts from the total does not give the live count. Nothing is deleted. A superseded row shows the run that replaced it.

The row-per-run table is in the archive as `results/tables/app_allruns.tex`, written by `code/make_tables.py`: run id, tag, model, arm, denominator, the three rates, the output-token cap and the status, with a retired row naming the run that replaced it. We point at it instead of setting 631 rows of lookup in type, because it restates rows the archive already ships machine-readably. One thing it does that no other file does: on a live row the three rates are counted from that run's per-item file under the convention the rest of the paper uses, while a retired row carries the aggregate `runs.jsonl` recorded at ingest, which counts only the per-item rows that wrote the key, so a retired row and the row that replaced it can differ by nothing but that denominator. The recount moves a printed cell on 53 of the 136 live rows, and `results/split_audit.json` lists every one.

Per-item raw results. The archive's per-item table gives all 344 items with every model's outcome on all three metrics. This is the table to check a claim against: the aggregate numbers in the main text are sums over these rows, and anyone with the released items and labels can regenerate any of them, with one condition that is easy to miss. The denominators are not the row counts: a mechanism

number excludes items whose gold label stops at a family node, and the file has to say which those are. Most files do say, through a `gold_is_family` key the harness grew part-way through the project; 210 (file, metric) pairs in the archive predate it, 52 of them behind a live run and the rest behind superseded scorings that back no printed number. `results/flag_gaps.tsv` lists all of them with that status, and `code/make_tables.py` generates it. Where the key is absent, recomputing a rate as correct-over-rows gives a lower number than we print: on `flat__opus5__train__n258__cbs430.jsonl` it is 0.081 over 258 rows against the 0.088 over 239 this paper reports. The authoritative denominator is the `*_n` field stored beside every rate in `results/runs.jsonl`, which is what every number here uses. One trap deserves naming, because we walked into it while checking this: a superseded scoring and the live one that replaced it can differ only by a `__rs3` suffix in the filename, identical row for row, with the key present in the live copy and absent in the other. Join on the run record, not on the filename. Each model cell holds three slots, printed in the header as `fms`: `f` appears when the cause family is correct, `m` when the exact mechanism is, `s` when the fix site is, and the slot is blank otherwise. Read that way the table makes two things visible that no aggregate does: items where a model points at the right line while naming the wrong mechanism (a cell whose only mark is `s`) are common, and items that are blank for every model cluster by repository rather than by cause family.

That table is `results/tables/app_peritem.tex` in the archive, one row per item and one `fms` cell per model, built from the per-item run files under `ideas/I0021-perfdiag-decision-graph/probe/runs/`. Both files are what to read an item off; 344 rows of marks print nothing an aggregate in this paper does not, so they are not set in type here.

Train–dev comparison. Every number in this paper pools 258 train and 86 dev items, and the 32-label set, the decision graph and the few-shot demonstrations were all derived from train. Three quarters of the denominator is therefore the pool the taxonomy was induced from, so we split the main cell. 5 of 6 models score lower on dev than on train, by −0.155 to 0.004 on cause family, and 2 of those drops reach $p \leq 0.05$ nominally on a two-sided Fisher exact test (both the two weakest). Claude Haiku 4.5 falls from 0.376 to 0.221 at $p = 0.0085$, which clears its BH threshold of 0.0099 at rank 37, so we report it as a rejection under the whole-paper correction. Llama-4-Maverick falls from 0.349 to 0.221 at $p = 0.032$, which does not clear its threshold of 0.015 at rank 56, so we report the direction and not a rejection. The test is unpaired because the two splits share no items; a paired test here would treat two independent samples as one and return a smaller and wrong p. They do share inputs: 4 dev items have an input byte-identical to a train item and 1 of them the same gold label (Appendix F), which is a reason this dev column is not a clean out-of-sample reading, not a reason to pair the test. Two readings fit and this design does not separate them: the taxonomy fits the items it was induced from, or 86 items is too few to tell.

The exact-mechanism axis is where this matters most, and it does not read the same way. 4 of 6 models are lower on dev there and 2 of those reach $p \leq 0.05$ nominally, two of them, including the strongest: Claude Opus 5 falls from 0.552 to 0.417 ($p = 0.045$ against a threshold of 0.017, so we report the direction and not a rejection) and Claude Haiku 4.5 from 0.243 to 0.125 ($p = 0.034$ against a threshold of 0.016, so we report the direction and not a rejection). So what we can say is that the pooled number is the more favourable of the two, and that on cause family the models it favours most are the weakest ones while on exact mechanism the model it favours most is the strongest one, which is also the model that wrote the key on 324 of 344 items. A user reporting on test should expect the dev-like figure rather than the pooled one, and §2 prints the pooled one. `code/split_robust.py` writes `results/split_robust.json`.

E Per-repository and per-label-source breakdown

Table 11 gives cause-family accuracy by repository and label source, and Table 12 gives all three metrics for all six models. Difficulty varies with the repository and with the label source, and so does the floor, so the widest margin over it is not where accuracy is highest; the paragraphs below take those in turn, and none of them fits in the main text.

vLLM dominates the corpus (147 of the 258 train items) but it is *not* where accuracy is highest: it is the best repository for only 2 of the six models, and Megatron-LM is best for 3. Repository familiarity is therefore at most a partial explanation of the scores: the most-written-about repository

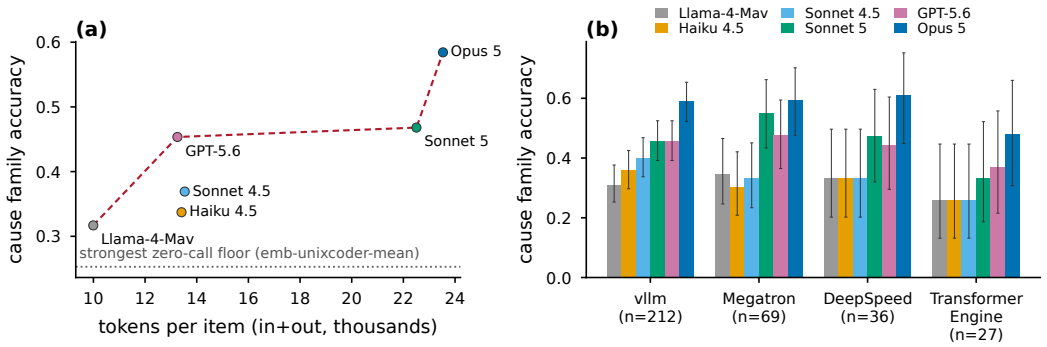

Figure 6: **(a)** Cost and quality on 344 items. Dashed, the Pareto front; dotted, the zero-call floor Table 1 prints (`emb-unixcoder-mean`, 0.253), which costs nothing. We draw that front through point estimates and plot no interval here, so its depth counts which models are not dominated on these two axes and does not rank them: the step from GPT-5.6 to Claude Sonnet 5 is inside both models' cause-family intervals in Table 1. §6.1 cites the depth with that restriction, and panel (b) refuses the same kind of reading for repositories. Tokens are input plus output per item as the API reports them, except GPT-5.6, whose `inputTokens` field is a single digit regardless of request size and whose input is therefore estimated at request bytes over four; without that estimate it plots at almost zero input cost and the front looks one point shallower. **(b)** Cause-family accuracy per repository, six models per group, item count under each repository, Wilson interval on every bar. The ordering differs from model to model and within a model no two repositories separate at 95%, so the per-repository ranking is not a finding; the one pattern every model shares is TransformerEngine, the smallest pool, last. This is the picture behind Table 11.

is the best one for 2 models and not for the rest, so a familiarity account has to explain the models it does not cover, and the tendency it does describe stays in the table. What is consistent is the bottom: TransformerEngine is the worst repository for all 6 models. It is the smallest subset (27 eval items) and the most kernel-level, so both a familiarity reading and a difficulty reading fit it. We cannot separate the two with what we have and say so in the limitations rather than picking either.

The floor is not flat across repositories, and that changes which repository looks hard. Table 12 carries no floor row. Broken out per repository from the same leave-one-repository-out folds that give the 0.238 char-n-gram family floor (Table 8), Megatron-LM carries the highest floor of the four and only the third largest margin over it, whether the margin is taken over the six models or over Opus 5 alone, and vLLM carries the largest. The ordering belongs to the fold regime rather than to the repositories: under the within-repository regimes in `results/floor_within.json` Megatron-LM has the lowest floor of the four and the margin ordering reverses. The regime this paper prints everywhere is the cross-repository one, so that is the one Table 8 decomposes, and we recommend no repository as a harder subset with a cleaner floor: the recommendation would flip with the regime.

Corpus-labelled and panel-labelled items are not equally hard, and the gap is not uniform across models. We report both columns everywhere and never pool them. Table 9 gives every split's composition on both dimensions, which is what a later user reporting a test number needs and what no sentence in this paper carried until we tabulated it: the panel share is 6% on the 344 items scored here and 88% on test, the vLLM share 62% against 87%, and dev is already part of the way to test on repository (76% vLLM) while staying with train on label source. So the test split is 86 items, 76 of them panel-labelled (88%) and 75 of them vLLM (87%); anyone reporting a test number must report the label-source split and the repository mix alongside it, and must report it twice, once on all 86 items and once on the 75 items whose answer key no author has read, a subset that is not the 75 vLLM items just quoted, because the label audit of Appendix I read 11 of the 86. That subset is the last column of Table 9 and it is composed like the full 86 on both dimensions, so the two numbers a user reports can be read side by side. The shift is in label provenance and in repository, not only in time. Tier is a third axis that shift runs along: test is 49% tier A (42 of 86) against 35% (122 of 344) on the scored pool, and tier A is the easier stratum, so a test number is optimistic on that axis too (`results/panel_eligibility.json`, Appendix A). Against the 344 items scored

Table 8: The 0.238 char-*n*-gram family floor broken out per repository, on the same leave-one-repository-out folds, beside cause-family accuracy on the same items. *cause* is the mean over the six models of Table 1, then Opus 5 alone; *– floor* is that minus the floor for that repository. Rows are ordered by the mean margin. The item counts are the same 344 scored items split four ways. `code/floor_repo.py` writes it from `results/runs.jsonl` and `results/floor_within.json`, and stops if the pooled cross-repository point estimate stops matching 0.238, the char-*n*-gram arm. Table 1's cause floor is the stronger pretrained-encoder arm `emb-unixcoder-mean`, 0.253 (Appendix G.5); this decomposition uses the char-*n*-gram arm, while Figure 1 draws each column's strongest zero-call arm.

			six models, mean		Claude Opus 5	
Repository	*n*	floor	cause	– floor	cause	– floor
vLLM	212	0.222	0.429	0.208	0.590	0.368
DeepSpeed	36	0.250	0.421	0.171	0.611	0.361
Megatron-LM	69	0.290	0.435	0.145	0.594	0.304
TransformerEngine	27	0.222	0.327	0.105	0.481	0.259

here, which is what any test number will be compared with, the vLLM share rises from 62% to 87% and TransformerEngine falls from 27 items to 2. Against dev alone the move is smaller and TransformerEngine does not move at all (2 items in dev, 2 in test), so the repository shift belongs to the comparison against the scored pool and not to the dev boundary. 50 of the 76 panel items in test are the freshest items in the benchmark, which have no card label available at all, so the provenance shift is partly a consequence of the time split and not independent of it. With DeepSpeed down to 1 and TransformerEngine to 2 items in test, only the vLLM cell is large enough to carry a per-repository test number at all, and `results/split_profile.json` gives the widest Wilson width each test cell admits under `test_per_repo_estimability`.

Two label-source counts appear in this paper and they are not the same count. The release-level split is 334 corpus-labelled and 96 panel-labelled over all 430 items, and that is the split §6.1 asks a test-split report to carry. Table 11 is computed on train+dev only, where the same split is 324 and 20. A reader looking in Table 11 for the 96 panel items will not find them: most of them are in the test split, which is why that table's panel column is the small one, and at that size its per-model values carry the interval given in its caption and cannot be compared across models. The 96-item panel population is measurable only once the test split is scored.

Table 9: Composition of every split on both dimensions the answer key varies along: label source and repository. Cells are item counts with the percentage of that column in parentheses. *train+dev* is the pool this paper measures every number on, *test* is the held-out split, and *test unread* is the 75 test items whose answer key no author has read (Appendix I). This table describes the test split, it does not score it: no model output on a test item exists. `code/split_profile.py` writes it from the frozen split file and the item census.

	train	dev	train+dev	test	test unread
Items	258	86	344	86	75
Label source					
corpus-labelled	242 (94)	82 (95)	324 (94)	10 (12)	10 (13)
panel-labelled	16 (6)	4 (5)	20 (6)	76 (88)	65 (87)
Repository					
vLLM	147 (57)	65 (76)	212 (62)	75 (87)	65 (87)
Megatron-LM	53 (21)	16 (19)	69 (20)	8 (9)	8 (11)
DeepSpeed	33 (13)	3 (3)	36 (10)	1 (1)	1 (1)
TransformerEngine	25 (10)	2 (2)	27 (8)	2 (2)	1 (1)

Agreement among evaluated models. Of the 344 items, 36 have all six models naming the correct cause family, 77 have none of them doing so, and 231 split the models. That shape says the benchmark discriminates: a majority of items are neither free nor impossible, which is what makes

Table 10: Cause-family accuracy per cause family per model, on 344 items. Columns are Llama-4-Maverick, Haiku 4.5, Sonnet 4.5, Sonnet 5, GPT-5.6, Opus 5.

Cause family	n	LLA	HAI	S4.5	S5	GPT	OP5
`expressed-intent-not-reached`	87	0.207	0.414	0.414	0.494	0.540	0.655
`steady-state-path-carries-avoidable-work`	63	0.794	0.587	0.714	0.730	0.635	0.683
`resource-lifetime-outlives-its-phase`	31	0.129	0.194	0.161	0.323	0.387	0.355
`compiled-region-does-not-cover-the-work`	30	0.167	0.167	0.233	0.300	0.300	0.367
`hardware-capability-misjudged`	30	0.333	0.300	0.267	0.467	0.500	0.733
`capacity-accounting-wrong-quantity`	29	0.103	0.241	0.207	0.517	0.310	0.517
`async-overlap-ordering-wrong`	19	0.474	0.158	0.211	0.211	0.316	0.579
`operand-form-mismatched-to-kernel`	17	0.235	0.294	0.588	0.529	0.471	0.588
`ranks-disagree-on-the-rendezvous`	16	0.000	0.188	0.062	0.250	0.125	0.688
`collective-scope-or-placement-wrong`	11	0.545	0.273	0.273	0.364	0.636	0.545
`recorded-decision-diverges-from-runtime`	11	0.000	0.182	0.182	0.273	0.091	0.364

Table 11: Cause-family accuracy by repository and by label source. Item counts: vLLM 212, Megatron-LM 69, DeepSpeed 36, TransformerEngine 27; corpus-labelled 324, panel-labelled 20. Corpus and panel labels are never pooled. The panel column has only 20 items in train+dev, because most panel-labelled items fall in the test split, so per-model panel estimates here carry intervals roughly ±0.20 and should not be compared across models.

Model	VLLM	MEGATRON	DEEPSPEED	TE	corpus	panel
Llama-4-Maverick	0.311	0.348	0.333	0.259	0.321	0.250
Claude Haiku 4.5	0.358	0.304	0.333	0.259	0.321	0.600
Claude Sonnet 4.5	0.401	0.333	0.333	0.259	0.358	0.550
Claude Sonnet 5	0.458	0.551	0.472	0.333	0.466	0.500
GPT-5.6	0.458	0.478	0.444	0.370	0.451	0.500
Claude Opus 5	0.590	0.594	0.611	0.481	0.580	0.650

model-to-model differences measurable rather than noise. It also locates the headroom. A hard core of 77 items, 22% of the pool, is named correctly at the family level by no model we tested. Table 13 shows one item of each kind with every model's answer.

Table 12: Every metric, every repository, every model, on 344 items: *fam* is cause family, *mech* exact mechanism, *site* line-level fix site. Cells are rounded to two decimals because three do not fit across this many columns: the *fam* cells to three decimals are the repository columns of Table 11, and the per-item outcomes every cell here is a sum over are in `results/tables/app_peritem.tex`.

		VLLM			MEGATRON			DEEPSPEED			TE		
Model	*n*	fam	mech	site	fam	mech	site	fam	mech	site	fam	mech	site
Llama-4-Maverick	344	0.31	0.17	0.30	0.35	0.16	0.35	0.33	0.11	0.19	0.26	0.15	0.41
Claude Haiku 4.5	344	0.36	0.20	0.37	0.30	0.19	0.45	0.33	0.19	0.42	0.26	0.19	0.56
Claude Sonnet 4.5	344	0.40	0.24	0.44	0.33	0.22	0.51	0.33	0.22	0.39	0.26	0.11	0.44
Claude Sonnet 5	344	0.46	0.32	0.41	0.55	0.42	0.48	0.47	0.31	0.33	0.33	0.22	0.44
GPT-5.6	344	0.46	0.28	0.42	0.48	0.33	0.54	0.44	0.33	0.36	0.37	0.33	0.56
Claude Opus 5	344	0.59	0.44	0.56	0.59	0.54	0.61	0.61	0.58	0.53	0.48	0.37	0.63

Table 13: One item of each agreement type, with each model's chosen mechanism. Answers marked *same family* earn family credit but not exact-mechanism credit.

Item	true mechanism / model	answer
all six models get the family right: `DeepSpeed@194053bd5894->ad661b8e3504`		
	truth	`per-step-device-to-host-readback`
	Llama-4-Maverick	`invariant-setup-hoisted-or-memoized` (same family)
	Claude Haiku 4.5	`per-step-device-to-host-readback`
	Claude Sonnet 4.5	`per-step-device-to-host-readback`
	Claude Sonnet 5	`per-step-device-to-host-readback`
	GPT-5.6	`per-step-device-to-host-readback`
	Claude Opus 5	`per-step-device-to-host-readback`
no model gets the family right: `DeepSpeed@7b5b06602d59->c502d3ac14ae`		
	truth	`gate-reads-wrong-operand`
	Llama-4-Maverick	`invariant-setup-hoisted-or-memoized` (wrong family)
	Claude Haiku 4.5	`per-step-device-to-host-readback` (wrong family)
	Claude Sonnet 4.5	`operand-form-normalized-before-kernel` (wrong family)
	Claude Sonnet 5	`per-element-loop-batched-or-folded` (wrong family)
	GPT-5.6	`persistent-buffer-binding-or-extent-wrong` (wrong family)
	Claude Opus 5	`operand-form-normalized-before-kernel` (wrong family)
models disagree: `DeepSpeed@0b224edcf7d8->5b314f4e6b87`		
	truth	`callee-binding-misresolved`
	Llama-4-Maverick	`invariant-setup-hoisted-or-memoized` (wrong family)
	Claude Haiku 4.5	`callee-binding-misresolved`
	Claude Sonnet 4.5	`invariant-setup-hoisted-or-memoized` (wrong family)
	Claude Sonnet 5	`callee-binding-misresolved`
	GPT-5.6	`hook-and-callback-order-vs-state` (wrong family)
	Claude Opus 5	`hook-and-callback-order-vs-state` (wrong family)

F Attribution provenance and model accuracy

266 of the 430 items (62%) carry an attribution this work produced rather than one the upstream project stated: 185 by SZZ line-blame from the fix's changed lines (Śliwerski et al., 2005) and 81 by reading the two diffs. The remaining 164 are tier A: 109 where the fix message names the inducing SHA, 29 reverts, 25 maintainer bisections and 1 CI attribution. SZZ's precision has been measured against developer-informed oracles and is well below one (da Costa et al., 2017; Rosa et al., 2021), so the tier B half cannot be assumed as clean as the tier A half, and a reader who wants only developer-attributed evidence should read the tier A column.

Table 16 scores each model on each tier separately. Every model does better on tier A. On cause family the gap runs from +0.068 to +0.228 and reaches $p < 0.05$ for 2 of the six; on fix line it

runs from +0.165 to +0.247 and is significant for all 6. The one cause-family test that is nominally significant without clearing its threshold sits at $p = 0.018$ against 0.012, so we report the direction and not a rejection. Each of the fix-line tests clears its Benjamini–Hochberg threshold, the weakest at $p = 0.0033$ against 0.0086, so we report it as a rejection under the whole-paper correction. The fix-line gap is larger, is significant everywhere, and one account of it is mechanical. When SZZ picks the wrong inducing commit, the item's diff is not the diff that caused the regression, so the lines the model is scored against (the lines the human fix touched) need not correspond to anything in what the model was shown. Cause family degrades less because the mechanism can still be right even when the specific commit is not, which is itself evidence that the two metrics fail for different reasons (§4.1). We cannot show that account is what produces the gap, and two measurements cap how much of it that account can carry. The tiers are not random halves of one population: the median tier B inducing commit changes 214 lines against 123 for tier A, the diff the model saw was clipped at the 900-line cap for 58 of the 222 tier B items against 24 of the 122 tier A ones, and standardising the comparison on quartiles of inducing-diff size shrinks Opus 5's fix-line gap from 0.197 to 0.148 and its cause-family gap from 0.072 to 0.034. Dropping the 22 reverts from tier A, where the fix diff is the inverse of the diff the model was shown, leaves 0.180 and 0.031. Inside tier B the direction runs against the account: the 153 SZZ-attributed scored items score above the 69 items paired by reading both diffs, for 5 of the six models on cause family and 5 of the six on fix line, and those two subsets differ in diff size as well. No comparison in this appendix separates attribution error from item difficulty, so the tier gap has to be read as a joint measure of both.

The tier split constrains how these numbers can be used, in two ways. Absolute numbers on the full 430 items are a blend of two populations and are lower than a developer-attributed-only benchmark would report: for the strongest model, 0.631 on tier A against 0.559 on tier B. And the model ordering is *mostly* but not entirely preserved: cause-family rank correlation between the two tiers is $\rho = 0.812$, and Claude Sonnet 4.5 moves from 4th to 6th. Two models separated by less than the tier A-versus-tier B gap should therefore be treated as tied unless the comparison is repeated within one tier.

A blind model audit bounds how often the link is right, and no human audit of it exists. The audit of Appendix I put one question to each sampled item, whether our mechanism label is what the fix addresses, so its 0.832 lower bound is a bound on the label. Its record carries 4 fields per item and 0 verdicts about the (inducing, fixing) pairing, although 6 of the 19 audited scored items are paired by SZZ and the worksheet showed the fix diff. The structural check a reader reaches for first separates nothing either: the fix diff touches a file the inducing commit touched for 344 of the 344 scored items, because file overlap is one of the entry criteria (Table 2). What needs no audit at all is the rule measured against the links the upstream projects state, and it answers a different question than the audit below does. Blaming the fix's changed lines at the fix's parent and taking the commit that owns most of them recovers the stated inducing commit for 88 of the 122 tier A scored items (72%), counting as misses the 7 whose fix only adds lines, and for 67 of 100 once the 22 reverts are dropped, where the fix diff is largely the inverse of the inducing diff and blame is at its easiest rather than infallible: it recovers 21 of those 22, missing 1, a revert whose fix deletes no line at all, so blame emits no candidate to rank. That is the agreement rate of one plain implementation of the rule on the items whose answer is known, and it reads tier B optimistically for two reasons: tier A is the easy case for blame, and a miss can be our implementation being more literal than the corpus channel, which had a model read the blame output. Run on tier B, the same plain rule reproduces the recorded inducing commit for 109 of the 153 SZZ-attributed scored items. `code/link_validity.py` computes every number in this paragraph except the revert recovery rate above, which `code/szz_mechanism.py` computes. The number a reader actually wants is the share of the 185 inferred links that are correct, and the four paragraphs that follow do not measure it: their population is the 153 SZZ-attributed items of the 222 scored tier B ones, which leaves out both the 69 diff-read items beside them and every tier B item in test.

Blind model adjudication of inducing-to-fixing links. A manual audit of a tier B sample is the check this construction most obviously needs. This is not that. The adjudicators here are language models, and the 19-item human record of Appendix I is a different measurement, about the mechanism label rather than the link; the two are never pooled and should not be read side by side. What we ran is a three-arm blind audit over 395 worksheet rows and 1,580 model calls (`code/link_audit.py`). Each row shows two diffs and asks one question, whether the earlier

commit contains what the later one repairs. The prompt is byte-identical across arms, both commit messages are withheld so that a message naming the inducing SHA cannot give tier A away, and the verdict is the majority of 3 adjudicators, none of them an evaluated model of Table 1; GPT-5.6, which is evaluated, runs as a separate cross-vendor column and never enters the majority. That independence covers the adjudicators and stops there: the attribution pass never recorded which model inferred the links being judged (Appendix A), so we cannot rule out that a model of Table 1 wrote the pairing an independent panel is here confirming. Tier A, where the project states the link, is the positive control and fixes the sensitivity: accept 118/122 = 0.967, and 96/100 once the reverts are dropped. The arm under test is not all of tier B, and the population has to be named before the number is read: `code/link_audit.py` admits a tier B row only when the recorded attribution method is SZZ, so the audited arm is the 153 SZZ-attributed items of the 222 scored tier B ones, and the 69 tier B items paired by reading both diffs were never put to an adjudicator. That remainder is not a random slice of what is left: the SZZ-attributed items score *above* the diff-read ones for 5 of the six models on cause family and 5 of the six on fix line, so every accept rate below covers the higher-scoring part of tier B and reading it as a tier B number biases that number up. On that arm, 145/153 = 0.948 [0.900, 0.973]. The placebo arm pairs each fix with an R-hotfile decoy and fixes the false-accept rate, 1/120 = 0.008. Rogan–Gladen then corrects that SZZ-attributed arm, 153 of the 222 scored tier B items, to 0.980 [0.930, 1.000] over 10,000 row bootstraps, and to [0.932, 1.000] when the bootstrap resamples items rather than rows, which is the right unit because the placebo arm re-uses the audited arms' fix commits: the interval printed beside 0.980 everywhere in this paper is the row bootstrap, and the item-clustered one is the interval to hold against it. Table 14 gives all three arms for every adjudicator.

Sources of upward bias in the link estimate. Four things inside the audit push it up, a fifth pushes it down, and a sixth, which arm of tier B the audit covers at all, sits outside the audit and pushes up hardest. First, what is judged is plausibility read off two diffs, which is weaker than the link being correct: a commit can plausibly contain the repaired thing without being the commit that introduced the regression. Second, tier A over-represents the easy positives, since 22 of its 122 rows are reverts and the rest lean on links a project named in a message, so the sensitivity is too high, and too high a sensitivity raises the quotient. Third, an R-hotfile decoy only shares a file with the fix, whereas the commit SZZ actually mistakes for the culprit is the runner-up blame candidate, so 0.008 is a floor on the false-accept rate and the correction is biased up. That stronger arm is specified and not run: 85 tier B items have a runner-up candidate, and 0 of the 340 calls it needs are in the released cache. Fourth, 22% of the bootstrap draws put the uncorrected quotient above 1.0, with an unclamped upper limit of 1.030, so the design cannot separate this value from 1.0 and the printed upper limit is the clamp. Fifth, and in the other direction, the 11 rows with no majority at all are scored as rejects, which lowers the quotient rather than raising it: dropping them gives 0.990 against the reported 0.980, so on this axis the reported value is the conservative one. Sixth is the coverage above: the 69 diff-read tier B items are outside the arm entirely, so 0.980 is a statement about the 153 SZZ-attributed links, and carrying it to the 222 scored tier B items as a whole extrapolates from the part that scores higher to the part nobody audited. The correction itself is nearly an identity. Sensitivity is 0.967 and the false-accept rate is 0.008, so the Rogan–Gladen denominator is close to one and the step from 0.948 to 0.980 rescales by a factor barely above one. That buys only the miscalibration the tier A and placebo arms measure, and those two arms are the ones we construct; leniency inside the arm under test passes through it unchanged, so the 13/16 accept rate below is a number to hold against 0.980.

Sensitivity to the weakest adjudicator. Read alone, on the same 153-item SZZ-attributed arm, the three adjudicators correct to 0.997, 0.968 and 0.814, and the cross-vendor column to 1.000, which is the clamp and not a measurement: its raw quotient is 1.010. The lowest is Pixtral Large at 0.814 [0.681, 0.966], and that is what we report as the lower bound, because a majority of 3 averages a weak adjudicator away. Pixtral accepts 84 of the 107 tier A rows it answered, a sensitivity of 0.785 against 0.984 for Sonnet 4.6, so on its own reading the audit is much less able to recognise a link it should accept. It also returned prose instead of calling the verdict tool on 43 rows, so 43 rows rest on two votes and 2 of those two votes disagree. The missingness is not arm-dependent (three-arm permutation test, $p = 0.84$; no pairwise Fisher test between arms comes close to rejecting), but it is length-dependent: the rows it skipped have a median prompt of 26,135 characters against 17,762 ($p = 0.034$, which does not clear its Benjamini–Hochberg threshold of 0.015), so the rows carried by two votes are the longest, and by that proxy plausibly the hardest, items, on a point estimate we

do not defend as a rejection. Backfilling the missing votes from the cross-vendor column moves no arm's count at all, although that reading is contaminated by construction.

Comparison with SZZ replication. The two numbers measure different things and a reader will otherwise collide them. The replication rate asks whether *our literal re-implementation* of blame ranks the recorded commit first; the accept rate asks whether the recorded pair holds up when the two diffs are read. Both can hold at once, because blame can rank the true commit second. Our plain rule ranks the recorded commit first on 109 of the tier B items and the adjudicators accept 105 of those (the same rule reproduces 0.712 of the tier B SZZ links overall). On the 44 items where it does not rank the recorded commit first, the adjudicators still accept 40 (0.909), and on the 16 where blame never reaches that commit at all they accept 13. Either the corpus channel, which had a model read the blame output, was right where our re-implementation is too literal, or the adjudicators are lenient on exactly the items that matter most; this design cannot tell which, and both readings should be held. Two cheap diagnostics say at least that the adjudicators are not pattern-matching on filenames: the majority reports same-file-only reasoning on 0 tier A and 0 tier B rows against 8/120 placebo rows, and answers "cannot tell" on 1 tier A and 1 tier B rows.

Table 14: The blind link audit of `code/link_audit.py`, over 395 worksheet rows and 1,580 model calls. Upper block: the three arms, with a Wilson interval on the majority accept rate, and the rates at which the majority answers "cannot tell" or reports that its only reason to link the pair is a shared file. The tier B arm is the 153 SZZ-attributed items of the 222 scored tier B ones, not tier B as a whole: the 69 items paired by reading both diffs are unaudited, so every tier B cell here reads that tier optimistically. Lower block: each adjudicator alone, with its own Rogan–Gladen correction and a 10,000-resample row bootstrap, which is the unit the text argues against; resampling items rather than rows is the right one. † GPT-5.6 is an evaluated model of Table 1, so it is a cross-vendor sensitivity column and is not part of the majority. The adjudicators are language models, not people.

Arm	n	accept	95% CI	cannot tell	same file
tier A, link stated upstream	122	0.967	[0.919, 0.987]	0.008	0.000
tier B, SZZ-attributed of 222 scored	153	0.948	[0.900, 0.973]	0.007	0.000
placebo, R-hotfile decoy	120	0.008	[0.002, 0.046]	0.000	0.067
tier A, no reverts	100	0.960	[0.902, 0.984]	–	–

Adjudicator	answered	tier A	tier B	placebo	corrected [95% CI]
Claude Sonnet 4.6	395	0.984	0.980	0.067	0.997 [0.962, 1.000]
Claude Opus 4.8	395	0.959	0.928	0.000	0.968 [0.912, 1.000]
Pixtral Large	352	0.785	0.650	0.056	0.814 [0.681, 0.966]
GPT-5.6†	395	0.984	0.994	0.033	1.000 [0.986, 1.000]
Majority of 3	395	0.967	0.948	0.008	0.980 [0.930, 1.000]

SZZ errors by mechanism. The standard objection to SZZ names three mechanisms, omission, configuration change and architectural interaction, so we split that tier A agreement rate by the item's gold cause family instead of pooling it (Table 15). Each cell carries two readings: the upstream-stated commit is the top-ranked blame candidate, and it is anywhere in the candidate set. The second is the one to hold against the released links, because the corpus's tier B SZZ channel had a model read the blame output rather than take the top candidate. Mapping the three words onto families by their taxonomy definitions, top-1 agreement is 1 of 5 on recorded decision diverges from runtime (any candidate 2; $p = 0.021$, against a Benjamini–Hochberg threshold of 0.013), 3 of 9 on the two architectural-interaction families (any candidate 7; $p = 0.014$, threshold 0.011) and 25 of 34 on expressed intent not reached (any candidate 27; $p > 0.99$), against 88 of 122 pooled and 103 on the any-candidate reading. On top-1 two of the three go the way that objection predicts and the configuration one goes against it. On the any-candidate reading, the one this paragraph privileges, all three sit below pooled: 40% latent, 78% architectural and 79% configuration against 84% pooled, so the one favourable cell of the top-1 reading is not favourable under the reading we prefer. That second column separates two failures the first one conflates. On async overlap ordering wrong, the lowest top-1 cell at 17%, the stated commit is among the blame candidates 5 of 6 times (83% against

84% pooled), so that cell is our tie-break ranking the right commit below another rather than blame failing to reach it, and the top-1 figure should not be read alone; the family blame really does not reach is recorded decision diverges from runtime, at 2 of 5. The omission mechanism has a structural reading that needs no ranking at all: 22 of the 344 scored items have a fix that deletes no product-code line, 21 of them no line in any file, so blame emits nothing; 7 of those are tier A and counted as misses above, 15 are tier B, and 7 of the tier B ones are recorded with SZZ as the attribution method, where that method name cannot be read literally.

Tier-B composition and reweighting. The families whose tier A top-1 rate is below the pooled rate carry 49 of the 153 tier B SZZ links (32%), not only the two that objection names. The largest single exposure is resource lifetime outlives its phase: 20 tier B SZZ links behind 2 of 4 tier A items, or 1 of 3 with reverts dropped, and the same family also holds 6 of the 22 add-only fixes, more than any other. Reweighting the tier A per-family rates by the tier B SZZ family mix gives 63% with no-revert weights, the reading that pool supports because it contains 0 reverts, against 68% with revert-inclusive weights: the mix effect is about 4.36 points either way, but the two levels sit 5.06 points apart and the revert-inclusive one is the optimistic of the two. One number cuts against the whole extrapolation and belongs here: the same plain rule's observed reproduction of the recorded link on those items is 71%, which is 8.60 points *above* the no-revert extrapolation, so the family mix does not transfer across tiers the way the reweighting assumes. Both point gaps here are differences of the unrounded rates, so neither equals the difference of the rounded percentages printed beside it. Nor is the spread itself established. The permutation test over these 11 families does not reject a constant rate ($p = 0.071$, 50,000 shuffles), although the asymptotic χ^2 on the same statistic does ($p = 0.037$); with a median cell of 6 items the permutation p is the one to read, and no single cell here should be read as an established mechanism effect. The three per-mechanism Fisher tests were fixed by that objection's own wording before the split, and all four p values enter the multiple-comparison family of Appendix H. None of this estimates the precision of the tier B links: it measures one plain re-implementation against the links upstream states, on the half where an answer exists. `code/szz_mechanism.py` computes every number in these two paragraphs.

Items that share an input. What the model sees is determined by the inducing commit, so two regressions traced to the same commit produce two items with byte-identical inputs and different answers. 45 of the 344 scored items fall in 21 such groups. A deterministic model can score at most the most common label within each group, which puts a structural ceiling of 0.956 on cause family over all 344 items and 0.952 on exact mechanism over the 311 of them whose gold label reaches a leaf, which is that column's own denominator (§2); dividing the mechanism loss by 344 instead gives 0.942, a ceiling for a column no table here scores. That is far above anything measured in this paper, so it changes no conclusion, but a reader comparing against 1.0 should know that 1.0 is not reachable. There is a second consequence, for the intervals rather than the ceiling, and it runs against us. A deterministic model gives one prediction per group, so outcomes inside a group are correlated: 15 of the 21 groups agree within themselves on the strongest model, mostly by getting every member wrong. Every Wilson interval on the full scored pool is computed on 344 independent items, and the smaller subsets that carry Wilson intervals elsewhere in this paper inherit the same correlation wherever one of these groups sits inside them. Under the worst case, perfect agreement within every group, the effective count is 320 and each interval should be about 4% wider. The measured agreement is not perfect, so the true inflation is smaller than that, and 4% does not change which comparisons separate; we report the bound rather than recompute every interval, and a reader who wants the recomputation has the group identifiers. We keep both items rather than dropping one: each records a real regression with a real fix, and choosing which to keep would be a judgement about which fix matters. A user who wants a single-answer-per-input pool can drop the 45 items with the released group identifiers.

The released split is not group-aware. The same group key over all 430 released items, not just the scored pool, finds 26 groups covering 59 items, and 9 of those groups straddle a split boundary over 23 items. `code/splits.py` cuts train, dev and test at two quantiles of fix date, and a date cut cannot respect a grouping on the inducing commit, so the boundaries fall inside groups. 8 of the 86 test items (9%) therefore have an input byte-identical to a released dev item, and 2 of those carry the same gold label, so anyone who tunes on dev and copies the answer across gets them right without diagnosing anything; the other 6 share the input and not the label, so they push a dev-tuned

Table 15: Tier A agreement of one plain SZZ re-implementation with the upstream-stated inducing commit, by gold cause family, on the 122 tier A scored items. *top-1* is the stated commit ranked first by blamed-line share; *any cand.* is the stated commit anywhere in the candidate set, the reading that matches the corpus's model-read-the-blame-output channel; *No revert* drops that row's reverts; *Tier B SZZ* counts how many of the 153 tier B SZZ links carry that family. Rows are ordered by top-1 rate, worst first; intervals are Clopper-Pearson. The permutation test over these 11 families does not reject a constant rate ($p = 0.071$), although the asymptotic χ^2 on the same statistic does ($p = 0.037$); with a median cell of 6 items the permutation p is the one to read, so no single cell here is an established mechanism effect.

Gold cause family	Tier A	top-1	95% CI	any cand.	No revert	Tier B SZZ
async overlap ordering wrong	6	1 (0.17)	[0.00,0.64]	5 (0.83)	1/6	12
recorded decision diverges from runtime	5	1 (0.20)	[0.01,0.72]	2 (0.40)	1/5	5
resource lifetime outlives its phase	4	2 (0.50)	[0.07,0.93]	2 (0.50)	1/3	20
collective scope or placement wrong	6	4 (0.67)	[0.22,0.96]	5 (0.83)	3/5	3
ranks disagree on the rendezvous	3	2 (0.67)	[0.09,0.99]	2 (0.67)	2/3	9
expressed intent not reached	34	25 (0.74)	[0.56,0.87]	27 (0.79)	15/23	34
compiled region does not cover the work	4	3 (0.75)	[0.19,0.99]	4 (1.00)	3/4	15
operand form mismatched to kernel	10	8 (0.80)	[0.44,0.97]	9 (0.90)	6/8	4
hardware capability misjudged	12	10 (0.83)	[0.52,0.98]	11 (0.92)	10/12	14
steady state path carries avoidable work	30	25 (0.83)	[0.65,0.94]	29 (0.97)	19/24	22
capacity accounting wrong quantity	8	7 (0.88)	[0.47,1.00]	7 (0.88)	6/7	15
Pooled	122	88 (0.72)	[0.63,0.80]	103 (0.84)	67/100	153

Table 16: Cause family and fix line by attribution tier, on 344 items (122 tier A, 222 tier B item-runs). Tier A is upstream-stated attribution, tier B is attribution this work inferred, mostly by SZZ. p is a two-sided Fisher exact test; the two tiers are disjoint item sets, so McNemar does not apply.

	cause family				fix line			
Model	tier A	tier B	Δ	p	tier A	tier B	Δ	p
Llama-4-Maverick	0.361	0.293	+0.068	0.23	0.434	0.234	+0.200	<0.001
Claude Haiku 4.5	0.385	0.311	+0.074	0.19	0.525	0.342	+0.183	0.0013
Claude Sonnet 4.5	0.516	0.288	+0.228	<0.001	0.607	0.360	+0.247	<0.001
Claude Sonnet 5	0.541	0.428	+0.113	0.055	0.549	0.347	+0.202	<0.001
GPT-5.6	0.541	0.405	+0.136	0.018	0.557	0.392	+0.165	0.0033
Claude Opus 5	0.631	0.559	+0.072	0.21	0.697	0.500	+0.197	<0.001

prediction the wrong way. 0 test items share an input with a train item. In the other direction 4 dev items have a byte-identical input in train, 1 of them with the same gold label, which is why Appendix D's train-against-dev paragraph says the two splits share no *items*: that is true of items, and the stronger reading, that they share no inputs, is false. Within test there are 3 further groups over 6 items, which caps within-test exact mechanism at 0.977 and cause family at 0.977 (the two coincide because all 86 test items have a gold label that reaches a leaf and the groups whose labels differ differ at family level too). The within-repository learned floor this paper prints comes from

folds `code/floor_within.py` keeps grouped by inducing commit, so that floor was not fitted on an item whose input it was scored on later. The released split files cannot do the same without being re-cut and we have not re-cut them. Nothing reported in this paper moves, because every number here is train or dev; what moves is what a later test number means, and a user who wants a group-clean split can cut one from the group key, which is `(repo, inducing_sha)` and is recoverable from the prefix of every released `pair_id`.

G Extended ablations and negative results

This appendix reports all evaluated variants, including unsuccessful ones and results affected by corrected instrumentation errors.

G.1 Output caps and invalid cells

Effects of increasing the output cap. Table 22 is the before-and-after. At a 4,096-token output cap the two strongest models truncated 50% to 52% of their *calls*. The denominator is calls and not items, which differ here because a truncated answer is retried: the two rates are 16 of 32 calls and 14 of 27 calls, so they must not be read as half of the items. Both come from 2 dedicated 20-item probes and not from the main cells, because the truncation counter did not exist when the main cells were first run. Sonnet 5's parse rate fell to 0.744, below the 0.90 gate, voiding the cell; Opus 5 recovered on a forced continuation turn, so the gate did not catch it and its reported score was a lower bound. At 32,000 tokens Opus 5 truncates nothing and Sonnet 5 truncates on 1 of 344 items, so the raised cap is nearly but not entirely non-binding. Opus 5's dev family accuracy *fell* from 0.550 to 0.523 when we raised the cap. That pre-raise number is the low-cap sampling scored by the *current* scorer, which is the only comparison the pair admits: the same sampling sits in the database under several scorings as the scorer changed, and an earlier one reads higher, so quoting it would put part of the scorer's history inside a number the text attributes to the cap. The cap was raised because a generation configuration must not bind on the length of an answer. The exception to the cap Setup declares is 50 of the 136 live cells, whose `config.max_tokens` is null, and it is wider than the prompt sweep. The 22 prompt-sweep cells of §4, the 20 variants plus the two flat baseline draws that print as one row of Table 25, are 22 of them, with 109 rows of that family null live and retired and 0 of their run summaries carrying the effective-configuration field the harness only grew later. The other 28 include all 10 rows of the `M3` ladder §5 plots, which is why Figure 5 does not have one cap. The remaining 5 live cells do record a value, below the declared cap: 3 zero-call floors, which make no model call at all, and 2 deliberate 4,096-token probes of the cap itself. We recover the cap two ways instead. Both search drivers pin it on the command line at 4,096, which is also the harness default (4,096); and the response cache, which keys on the request body and therefore stores the requested `maxTokens` of every distinct call, holds 3,868 distinct Claude Sonnet 4.5 diagnosis requests at that cap, 3,090 of them attributable to dev items, 0 of which stopped at the cap, longest answer 3,805 tokens. That cache is a set of distinct requests and not a call log: the sweep's per-item files record 3,195 calls over those cells, of which 134 come from the two pairs of cells that are the same draw and therefore share cache entries (`B0-baseline` with `S00-flat-baseline`, `B7-fulldefs` with `S9-flat-full-defs`), and the dev-attributed entries then run 29 above what that leaves, which is the slack in the (repository, author date) attribution key. One reading favours us and two run against us. Truncated responses are cached, 48 such entries at other caps, so 0 is evidence and not survivorship. Against us, the same cache shows the model hitting a cap whenever the cap is lower: 2 of 332 diagnosis calls at 2,600 tokens, all 332 of them on dev items of this benchmark; and outside the diagnosis prompt entirely, 46 of 610 calls, a rate of 0.075, at 1,200 tokens, the cap the relabelling, coverage-ablation and link-audit jobs use, and those jobs do produce numbers in this paper. What the cache establishes is therefore narrow: 4,096 did not bind these cells. It does not establish that this model cannot reach 4,096, because on this prompt family it never did.

The `M3` ladder with every row at one cap, and what that costs the non-monotonicity claim. Figure 5 draws 3 rows at the recovered 4,096 default beside 2 at the declared 32,000, and that split runs with the deficit ranking, so the 10.45 to 22.39 spread it shows is not a spread at one cap. Fixing this needs no new call: the retired default-cap draws of the two re-drawn rows are still in the database, so Table 18 puts all 6 rows at that one cap, re-scored by the current scorer on the same 67 dev items. The shape holds and the margin does not. Claude Sonnet 4.5 still carries the largest deficit, 13.43

Table 17: The oracle comparison of Figure 7 in full, on the 67-item dev pool. *Flat menu* is exact mechanism over the 59 leaf-gold items with all 32 labels offered; *Graph, family given* is the same items with the true family injected; *Ratio* is their quotient with a 10,000-resample paired-item bootstrap interval; *Routing* is how often the graph's second layer picks the correct family unaided, over all 67. Chance for the family-given column, uniform within the true family, is 0.294.

Model	Flat menu	Graph, family given	Ratio	95% CI	Routing
Claude Sonnet 4.5	0.271	**0.542**	2.00×	[1.44, 3.11]	0.224
Claude Opus 5	0.492	**0.763**	1.55×	[1.21, 2.09]	0.478

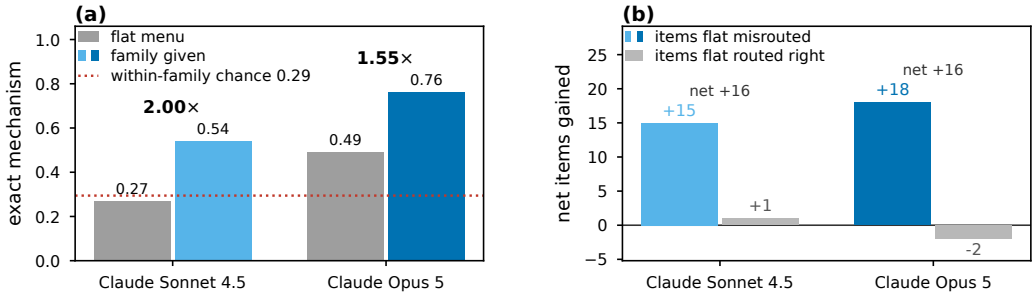

Figure 7: Handing the graph the correct cause family reverses the comparison. The reversal moves the family label and the menu length together, and no arm in this design separates them. Dev pool, exact mechanism over the 59 leaf-gold items. **(a)** Flat menu (all 32 labels) against family given (its 2 to 6 mechanisms): 2.00× and 1.55 ×; the dotted line is the within-family chance rate 0.294. **(b)** Net items gained, split by whether the flat call already routed the item right: the whole gain is on items it *misrouted* (15, 18). Where flat had the family right, the restricted menu changes nothing we can detect (+1 and −2 items on 26 and 39, exact McNemar p >0.99 and 0.69), but that null has almost no resolution: for Sonnet 4.5 no split of its 3 discordant items reaches 0.05, so this cell carries no resolution at all, and for Opus 5 the smallest difference it could have rejected is 6 of its 39 items, 0.154 accuracy, against contrasts of 0.271 to 0.542 and 0.492 to 0.763. On the misrouted half, which carries the whole gain, the family label and the shortened menu are collinear by construction. So the multiple is an upper bound on the value of the family label and not a measurement of it. Full table with routing and bootstrap intervals: Table 17.

points against 10.45 for Claude Opus 5, but it leads by 2.99 points, which is 2 items of 67, where the mixed-cap figure makes that lead 8.96. At this cap 4 of the 6 rows lose more than five points and 2 lose nothing, and the peak's own paired test does not clear the whole-paper correction: it is rank 81 of 186 and does not clear its threshold (p = 0.11 against a threshold of 0.022), the draw Table 25 prints as `S01-graph-baseline`. So §5 claims the ordering of the deficits and not the size of the gap between them. Two things in the table are not free. Claude Sonnet 5 has a row here and none in Figure 5 because at this cap its two cells fail both validity gates, parse 0.761 and 0.851 against 0.90 and unfinished 0.194 and 0.149 against 0.10, and the re-draw that repaired its main-table cells (0.744 to 0.953 on dev) never reached its ladder cells. We print the row because leaving the second model of Table 1 out of an argument about capability is worse than printing a bounded one, and the bound runs against us: the flat arm loses more answers to the gate than the graph arm does, so its deficit here is a lower bound. Second, we did not re-draw the 3 default-cap rows at 32,000. That is the run that would let the claim keep the strength Figure 5 suggests, and we report the reduced strength instead.

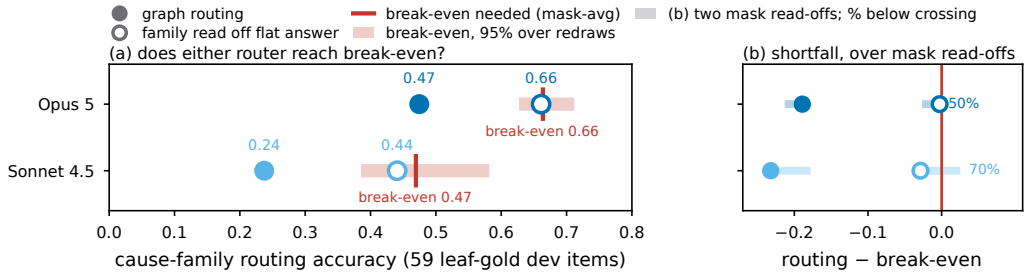

Figure 8: Whether the decision graph is worth walking is decided by routing accuracy, and neither candidate router reaches the break-even it would need, on the 59 leaf-gold dev items. **(a)** Per model: walking the graph layer by layer (filled) and reading the cause family off the flat-menu answer instead (hollow), against the break-even routing accuracy that would match the flat menu (red line). The threshold moves with the item-id-hash mask, so the line is mask-averaged and the band is its 95% interval over 10,000 redraws. **(b)** The same two routers as a signed shortfall, with the span between the two mask read-offs. The graph falls below its *whole* band (0.232 and 0.189 short), which is what §5 defends; the flat read-off falls short of both by 0.029 and 0.003, margins that change sign with the mask and sit below the crossing in 70% and 50% of redraws (the percentages in (b)), so we report no crossing for it either. Neither the sizes nor the ordering of the gaps is defended (Appendix G).

Table 18: The `M3` dev ladder with every row at one output cap, the 4,096 tokens the harness defaults to, recovered from the driver scripts rather than read off the run record. *deficit* is flat minus graph in cause-family accuracy points and *items* is the same difference in items out of 67. These are the draws Figure 5 shows for Llama-4-Maverick, Claude Haiku 4.5 and GPT-5.6 and the retired predecessors of the draws it shows for Claude Sonnet 4.5 and Claude Opus 5, all re-scored by the current scorer, which is the only scoring the pair admits. The dagger marks the one row that does not pass the validity gates at this cap and whose two accuracies are therefore lower bounds; the paragraph above says why it is printed anyway.

Model	flat	95% CI	graph	95% CI	deficit	items
Llama-4-Maverick	0.313	[0.215, 0.432]	0.328	[0.228, 0.447]	−1.49	−1
Claude Haiku 4.5	0.299	[0.202, 0.417]	0.299	[0.202, 0.417]	0.00	0
Claude Sonnet 4.5	0.418	[0.307, 0.537]	0.284	[0.190, 0.401]	13.43	9
Claude Sonnet 5†	0.388	[0.280, 0.508]	0.313	[0.215, 0.432]	7.46	5
GPT-5.6	0.463	[0.349, 0.581]	0.358	[0.254, 0.478]	10.45	7
Claude Opus 5	0.597	[0.477, 0.706]	0.493	[0.377, 0.609]	10.45	7

Table 19: The 11 cause families, the mechanisms each contains, how many of the 344 items fall in each, and Opus 5's cause-family accuracy on each. The names in the first column are short descriptions; the second column is the identifier the corpus gives that family with its layer prefix stripped, which is the same string Table 10 prints, so the two tables can be joined row by row rather than by position. The first sentence of each definition is in Table 6, and the definition in full is in the release. Over the 11 families Opus 5's accuracy runs from 0.355 to 0.733, so a headline number hides which branch a model fails on. Appendix E gives all six models.

Cause family	identifier in the corpus	mechs.	items	OPUS 5
Setting never takes effect	`expressed-intent-not-reached`	6	87	0.655
Avoidable work on the hot path	`steady-state-path-carries-avoidable-work`	5	63	0.683
Resource outlives its phase	`resource-lifetime-outlives-its-phase`	3	31	0.355
Work escapes the compiled region	`compiled-region-does-not-cover-the-work`	2	30	0.367
Wrong answer about the hardware	`hardware-capability-misjudged`	3	30	0.733
Wrong size or budget	`capacity-accounting-wrong-quantity`	2	29	0.517
Misplaced ordering between streams	`async-overlap-ordering-wrong`	3	19	0.579
Operand form wrong for the kernel	`operand-form-mismatched-to-kernel`	2	17	0.588
Ranks disagree at a rendezvous	`ranks-disagree-on-the-rendezvous`	2	16	0.688
Collective over the wrong group	`collective-scope-or-placement-wrong`	2	11	0.545
Stale decision replayed	`recorded-decision-diverges-from-runtime`	2	11	0.364

Table 20: Cause-family accuracy on train and dev, split by whether the two labelling passes agree on the family. The agreement number the paper prints elsewhere (0.794) is computed only on the 252 items that carry a second label, and on those placement is easier: all 6 of 6 models score lower where the passes disagree than where they agree. The 92 items with no second label are not a harder rung still, and the three strata are not a difficulty ordering: 5 of 6 models score higher there than where the passes disagree, and 1 of them (Claude Sonnet 4.5) higher there than where the passes agree. Only the first comparison bears on the agreement number, and it holds for every model, so we call that number a two-pass agreement rather than a ceiling and do not read the headroom it implies as headroom to human accuracy.

	both passes agree		passes disagree		no second label	
Model	*n*	family	*n*	family	*n*	family
Llama-4-Maverick	200	0.370	52	0.250	92	0.239
Claude Haiku 4.5	200	0.390	52	0.212	92	0.293
Claude Sonnet 4.5	200	0.365	52	0.327	92	0.402
Claude Sonnet 5	200	0.500	52	0.365	92	0.457
GPT-5.6	200	0.475	52	0.385	92	0.446
Claude Opus 5	200	0.665	52	0.423	92	0.500

Table 21: Detection under neutral framing, one row per model, on the re-run with the output cap matched across models. *real items* is how often the model claims a regression on a commit that has one; the three R- columns are the decoy strata, which contain none. The last column is the difference a reader would compute, given here so it is computed within a model rather than across two: every value in a row comes from the same model. An earlier version of this table had no model column and put one model's recall above another model's false-positive rates, which invited a subtraction that overstated discriminability by more than a factor of two.

Model	real items	R-CLEAN	R-HOTFILE	R-RANDOM	recall − clean
Claude Sonnet 4.5	0.802	0.521	0.549	0.468	0.281
GPT-5.6	0.430	0.129	0.226	0.175	0.301
Claude Opus 5	0.686	0.239	0.268	0.158	0.447

Table 22: Every run whose output cap was raised, one row each, keyed by run id. 148 rows; 6 rows are omitted because the run recorded neither a parse rate nor a family accuracy. Truncation is the fraction of calls that stopped because they hit the cap. The *cap* column is a recorded configuration value on the unstarred rows only; on the 120 starred ones it is the harness default. * the run record carries no output cap for 120 of the 148 rows; the value shown is the harness default recovered from the driver scripts by `code/sweep_cap_audit.py`, not a recorded value. Compare two rows only when the model, arm and split all match: those are the before-and-after pairs. An earlier version of this table iterated over supersession markers instead of runs, so the same run appeared up to 24 times, and it was set as a `tabular` inside a float, which silently dropped every row past the page bottom.

run	model	arm	split	cap	parse	unfin.	trunc.	out/call	family
E4072	GPT	flat	dev	4,096*	1.000	0.000	–	2,188	0.384
E8567	GPT	flat	dev	4,096*	1.000	0.000	–	2,031	0.430
E5151	GPT	flat	dev	4,096*	1.000	0.000	–	2,188	0.384
E1783	GPT	flat	dev	4,096*	1.000	0.000	–	2,031	0.430
E5472	GPT	flat	dev	4,096*	1.000	0.000	–	2,188	0.384
E8678	GPT	flat	dev	4,096*	1.000	0.000	–	2,031	0.430
E5287	GPT	flat	dev	4,096*	1.000	0.000	–	2,188	0.384
E9700	GPT	flat	dev	4,096*	1.000	0.000	–	2,031	0.430
E2952	GPT	flat	dev	32,000	1.000	0.000	0.000	2,043	0.442
E6575	GPT	flat	dev	32,000	1.000	0.000	0.000	2,095	0.465
E8608	GPT	flat	train	4,096*	1.000	0.000	–	2,004	0.484
E9621	GPT	flat	train	4,096*	1.000	0.000	–	2,012	0.492
E5559	GPT	flat	train	4,096*	1.000	0.000	–	2,004	0.484
E9780	GPT	flat	train	4,096*	1.000	0.000	–	2,012	0.492
E2394	GPT	flat	train	4,096*	1.000	0.000	–	2,004	0.484
E6069	GPT	flat	train	4,096*	1.000	0.000	–	2,012	0.492
E8612	GPT	flat	train	4,096*	1.000	0.000	–	2,004	0.484
E3763	GPT	flat	train	4,096*	1.000	0.000	–	2,012	0.492
E3604	GPT	flat	train	32,000	1.000	0.000	0.000	2,141	0.457
E5503	GPT	flat	train	32,000	1.000	0.000	0.000	2,030	0.508
E5319	HAI	flat	dev	4,096*	1.000	0.000	–	1,278	0.233
E7885	HAI	flat	dev	4,096*	1.000	0.000	–	1,278	0.233
E4027	HAI	flat	dev	4,096*	1.000	0.000	–	1,278	0.233
E5924	HAI	flat	dev	4,096*	1.000	0.000	–	1,278	0.233
E7905	HAI	flat	dev	32,000	1.000	0.000	0.000	1,265	0.221
E5362	HAI	flat	train	4,096*	1.000	0.000	–	1,256	0.376
E5993	HAI	flat	train	4,096*	1.000	0.000	–	1,256	0.376
E7604	HAI	flat	train	4,096*	1.000	0.000	–	1,256	0.376
E6072	HAI	flat	train	4,096*	1.000	0.000	–	1,256	0.376
E5539	HAI	flat	train	32,000	1.000	0.000	0.000	1,254	0.376
E5760	LLA	flat	dev	4,096*	0.988	0.000	–	261	0.198
E6587	LLA	flat	dev	4,096*	0.988	0.000	–	261	0.198
E5152	LLA	flat	dev	4,096*	0.988	0.000	–	261	0.198
E8423	LLA	flat	dev	4,096*	0.988	0.000	–	261	0.198
E6443	LLA	flat	dev	32,000	0.988	0.000	0.000	257	0.221
E3928	LLA	flat	train	4,096*	0.996	0.000	–	250	0.337
E5650	LLA	flat	train	4,096*	0.996	0.000	–	250	0.337
E8829	LLA	flat	train	4,096*	0.996	0.000	–	250	0.337
E4931	LLA	flat	train	4,096*	0.996	0.000	–	250	0.337
E4131	LLA	flat	train	32,000	0.996	0.000	0.000	244	0.349
E2911	OP5	flat	dev	4,096*	0.919	0.081	–	5,632	0.557
E5879	OP5	flat	dev	4,096*	0.940	0.060	–	5,544	0.635

continued on the next page

Table 22, continued

run	model	arm	split	cap	parse	unfin.	trunc.	out/call	family
E5016	op5	flat	dev	4,096*	1.000	0.000	–	2,962	0.407
E4689	op5	graph	dev	4,096*	0.955	0.045	–	4,257	1.000
E4601	op5	graph	dev	4,096*	0.970	0.030	–	11,993	0.508
E6428	op5	flat	dev	4,096*	0.919	0.070	–	5,487	0.562
E6023	op5	flat	dev	4,096*	0.919	0.081	–	5,632	0.557
E1267	op5	flat	dev	4,096*	0.940	0.060	–	5,544	0.635
E3767	op5	flat	dev	4,096*	1.000	0.000	–	2,962	0.407
E4962	op5	graph	dev	4,096*	0.955	0.045	–	4,257	1.000
E3434	op5	graph	dev	4,096*	0.970	0.030	–	11,993	0.508
E3235	op5	flat	dev	4,096*	0.919	0.070	–	5,487	0.562
E1246	op5	flat	dev	4,096*	0.919	0.081	–	5,632	0.550
E6010	op5	flat	dev	4,096*	0.940	0.060	–	5,544	0.625
E3778	op5	flat	dev	4,096*	1.000	0.000	–	2,962	0.407
E9694	op5	graph	dev	4,096*	0.955	0.045	–	4,257	0.985
E2722	op5	graph	dev	4,096*	0.970	0.030	–	11,993	0.508
E7124	op5	flat	dev	4,096*	0.919	0.070	–	5,487	0.549
E3482	op5	flat	dev	4,096*	0.940	0.060	–	5,544	0.625
E2676	op5	flat	dev	4,096*	1.000	0.000	–	2,962	0.407
E2190	op5	graph	dev	4,096*	0.970	0.030	–	11,993	0.508
E4152	op5	flat	dev	4,096*	0.919	0.070	–	5,487	0.549
E1254	op5	flat	dev	4,096*	0.919	0.081	–	5,632	0.550
E4086	op5	graph	dev	4,096*	0.955	0.045	–	4,257	0.985
E2852	op5	flat	dev	32,000	1.000	0.000	–	8,637	0.523
E4632	op5	flat	dev	32,000	1.000	0.000	0.000	8,369	0.612
E2679	op5	flat	dev	32,000	1.000	0.000	0.000	3,026	0.419
E1164	op5	graph	dev	32,000	1.000	0.000	0.000	5,135	1.000
E9651	op5	graph	dev	32,000	1.000	0.000	0.000	16,277	0.478
E4895	op5	flat	dev	32,000	1.000	0.000	0.000	10,041	0.558
E8190	op5	flat	train	4,096*	0.957	0.035	–	5,190	0.594
E4255	op5	flat	train	4,096*	1.000	0.000	–	3,910	0.306
E7864	op5	flat	train	4,096*	0.961	0.031	–	5,136	0.572
E7782	op5	flat	train	4,096*	0.957	0.035	–	5,190	0.594
E6246	op5	flat	train	4,096*	1.000	0.000	–	3,910	0.306
E9191	op5	flat	train	4,096*	0.961	0.031	–	5,136	0.572
E3021	op5	flat	train	4,096*	0.957	0.035	–	5,190	0.594
E3386	op5	flat	train	4,096*	1.000	0.000	–	3,910	0.306
E9604	op5	flat	train	4,096*	0.961	0.031	–	5,136	0.567
E6307	op5	flat	train	4,096*	0.957	0.035	–	5,190	0.594
E5041	op5	flat	train	4,096*	1.000	0.000	–	3,910	0.306
E7836	op5	flat	train	4,096*	0.961	0.031	–	5,136	0.567
E8745	op5	flat	train	32,000	0.992	0.008	0.000	7,621	0.609
E9961	op5	flat	train	32,000	1.000	0.000	0.000	4,407	0.314
E7201	op5	flat	train	32,000	0.996	0.004	0.000	7,648	0.599
E3230	s4.5	flat	dev	4,096*	1.000	0.000	–	1,364	0.326
E1217	s4.5	flat	dev	4,096*	1.000	0.000	–	1,387	0.418
E8617	s4.5	graph	dev	4,096*	1.000	0.000	–	1,399	1.000
E1713	s4.5	graph	dev	4,096*	1.000	0.000	–	4,010	0.284
E7296	s4.5	flat	dev	4,096*	1.000	0.000	–	1,151	0.337
E1832	s4.5	flat	dev	4,096*	1.000	0.000	–	1,397	0.360
E4402	s4.5	flat	dev	4,096*	1.000	0.000	–	1,364	0.326
E1645	s4.5	flat	dev	4,096*	1.000	0.000	–	1,387	0.418
E7696	s4.5	graph	dev	4,096*	1.000	0.000	–	1,399	1.000
E4129	s4.5	graph	dev	4,096*	1.000	0.000	–	4,010	0.284
E8057	s4.5	flat	dev	4,096*	1.000	0.000	–	1,151	0.337
E1711	s4.5	flat	dev	4,096*	1.000	0.000	–	1,397	0.360

continued on the next page

Table 22, continued

run	model	arm	split	cap	parse	unfin.	trunc.	out/call	family
E2791	s4.5	flat	dev	4,096*	1.000	0.000	–	1,364	0.326
E6747	s4.5	flat	dev	4,096*	1.000	0.000	–	1,387	0.418
E5265	s4.5	graph	dev	4,096*	1.000	0.000	–	1,399	1.000
E3388	s4.5	graph	dev	4,096*	1.000	0.000	–	4,010	0.284
E8857	s4.5	flat	dev	4,096*	1.000	0.000	–	1,151	0.337
E4439	s4.5	flat	dev	4,096*	1.000	0.000	–	1,397	0.360
E2860	s4.5	flat	dev	4,096*	1.000	0.000	–	1,364	0.326
E5509	s4.5	flat	dev	4,096*	1.000	0.000	–	1,387	0.418
E5423	s4.5	graph	dev	4,096*	1.000	0.000	–	1,399	1.000
E3212	s4.5	graph	dev	4,096*	1.000	0.000	–	4,010	0.284
E2822	s4.5	flat	dev	4,096*	1.000	0.000	–	1,151	0.337
E8620	s4.5	flat	dev	4,096*	1.000	0.000	–	1,397	0.360
E8107	s4.5	flat	dev	32,000	1.000	0.000	0.000	1,384	0.372
E3530	s4.5	flat	dev	32,000	1.000	0.000	0.000	1,392	0.448
E6525	s4.5	graph	dev	32,000	1.000	0.000	0.000	1,380	1.000
E6687	s4.5	graph	dev	32,000	0.985	0.015	0.000	4,057	0.227
E3188	s4.5	flat	dev	32,000	1.000	0.000	0.000	1,149	0.337
E7733	s4.5	flat	dev	32,000	1.000	0.000	0.000	1,399	0.337
E9660	s4.5	flat	train	4,096*	1.000	0.000	–	1,373	0.372
E8118	s4.5	flat	train	4,096*	1.000	0.000	–	1,162	0.260
E6654	s4.5	flat	train	4,096*	1.000	0.000	–	1,367	0.368
E6629	s4.5	flat	train	4,096*	1.000	0.000	–	1,373	0.372
E9661	s4.5	flat	train	4,096*	1.000	0.000	–	1,162	0.260
E9072	s4.5	flat	train	4,096*	1.000	0.000	–	1,367	0.368
E7282	s4.5	flat	train	4,096*	1.000	0.000	–	1,373	0.372
E8554	s4.5	flat	train	4,096*	1.000	0.000	–	1,162	0.260
E4386	s4.5	flat	train	4,096*	1.000	0.000	–	1,367	0.368
E3971	s4.5	flat	train	4,096*	1.000	0.000	–	1,373	0.372
E9628	s4.5	flat	train	4,096*	1.000	0.000	–	1,162	0.260
E1017	s4.5	flat	train	4,096*	1.000	0.000	–	1,367	0.368
E8390	s4.5	flat	train	32,000	1.000	0.000	0.000	1,380	0.368
E1647	s4.5	flat	train	32,000	1.000	0.000	0.000	1,161	0.236
E4277	s4.5	flat	train	32,000	1.000	0.000	0.000	1,372	0.384
E5364	s5	flat	dev	4,096*	0.744	0.233	–	5,347	0.455
E7024	s5	flat	dev	4,096*	0.800	0.175	–	4,662	0.485
E3798	s5	flat	dev	4,096*	0.800	0.175	–	4,662	0.485
E3207	s5	flat	dev	4,096*	0.744	0.233	–	5,347	0.455
E9558	s5	flat	dev	4,096*	0.800	0.175	–	4,662	0.471
E5808	s5	flat	dev	4,096*	0.744	0.233	–	5,347	0.435
E5143	s5	flat	dev	4,096*	0.800	0.175	–	4,662	0.471
E6193	s5	flat	dev	4,096*	0.744	0.233	–	5,347	0.435
E3352	s5	flat	dev	32,000	0.953	0.000	0.011	7,936	0.465
E1750	s5	flat	train	4,096*	0.891	0.097	–	4,608	0.472
E8021	s5	flat	train	4,096*	0.975	0.025	–	3,948	0.462
E4500	s5	flat	train	4,096*	0.891	0.097	–	4,608	0.472
E3955	s5	flat	train	4,096*	0.975	0.025	–	3,948	0.462
E6239	s5	flat	train	4,096*	0.891	0.097	–	4,608	0.464
E5728	s5	flat	train	4,096*	0.975	0.025	–	3,948	0.462
E4470	s5	flat	train	4,096*	0.891	0.097	–	4,608	0.464
E5107	s5	flat	train	4,096*	0.975	0.025	–	3,948	0.462
E2940	s5	flat	train	32,000	0.977	0.000	0.000	6,303	0.469

Table 23: Every arm crossed with every model that ran it and with the condition it ran under: 127 rows. The condition column is not decoration: rows that differ in it are not summable, and an earlier version of this table pooled them, which made the `graph` arm appear to beat `flat` for Opus 5 because the family-given upper-bound cells were averaged in. Rows reading *family given* are upper-bound experiments, not configurations one could deploy. Where two runs under the same condition share items, the aggregation guard refuses to add them and this table prints one row per run with its run id appended to the condition, rather than a sum that counts those items twice. That guard is why this table now has more rows than it used to: the earlier version summed by hand and bypassed it, which is how it printed an *n* of 411 for a pool of 344. *void* counts cells failing the parse/finish gate. The floor rows are the zero-call baselines, listed here so they can be read against the arms with the same scorer. A dash means the metric is undefined for that row by construction and not a measured zero, the same convention Table 1 uses: `freeform` answers are free text that was never placed on the label menu, so its family and mechanism cells are unmapped rather than wrong, and the mapped version of the same run is the `freeform_mapped` block; `symptom_only` is asked only for a symptom node, which can never equal a cause family or a leaf mechanism; and the one row whose menu offered cause families rather than leaf mechanisms cannot match a leaf either, so its mechanism cell is a dash here and in Table 25 rather than the 0.000 an earlier build printed.

Arm	model	condition	*n*	family	mech.	site	calls	status
`flat`								
	Llama-4-Maverick	cap 32,000	344	0.317	0.177	0.305	1.0	ok
	Llama-4-Maverick	cap default	67	0.313	0.186	0.224	1.0	ok
	Claude Haiku 4.5	cap 32,000	344	0.337	0.215	0.407	1.0	ok
	Claude Haiku 4.5	cap default	67	0.299	0.169	0.299	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E9963	67	0.448	0.271	0.358	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E7965	86	0.372	0.250	0.360	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E2525	258	0.368	0.247	0.477	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E8286	67	0.448	0.271	0.358	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E3455	67	0.418	0.237	0.358	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E5356	67	0.403	0.203	0.328	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E9214	67	0.373	0.203	0.343	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E4765	67	0.373	0.237	0.403	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E8446	67	0.418	0.305	0.343	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E6013	67	0.373	0.237	0.343	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E4749	67	0.448	0.271	0.358	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E3055	201	0.423	0.260	0.373	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E5771	67	0.343	–	0.373	1.0	ok

continued on the next page

Table 23, continued

Arm	model	condition	n	family	mech.	site	calls	status
	Claude Sonnet 4.5	cap 32,000, E5692	67	0.403	0.322	0.358	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E8980	67	0.373	0.220	0.418	1.0	ok
	Claude Sonnet 4.5	cap 32,000, E1891	67	0.373	0.237	0.403	1.0	ok
	Claude Sonnet 4.5	cap default, E7668	40	0.475	0.263	0.450	1.0	ok
	Claude Sonnet 4.5	cap default, E5817	67	0.418	0.237	0.373	1.0	ok
	Claude Sonnet 4.5	cap default, E3413	67	0.373	0.237	0.358	1.0	ok
	Claude Sonnet 4.5	cap default, E4566	67	0.403	0.237	0.343	1.0	ok
	Claude Sonnet 4.5	cap default, E7554	67	0.388	0.203	0.328	1.0	ok
	Claude Sonnet 4.5	cap default, E9917	67	0.358	0.237	0.358	1.0	ok
	Claude Sonnet 4.5	cap default, E6408	67	0.403	0.271	0.373	1.0	ok
	Claude Sonnet 4.5	cap default, E1184	67	0.373	0.271	0.343	1.0	ok
	Claude Sonnet 4.5	cap default, E7660	67	0.418	0.237	0.373	1.0	ok
	Claude Sonnet 4.5	cap default, E3574	201	0.388	0.203	0.378	1.0	ok
	Claude Sonnet 4.5	cap default, E8518	67	0.313	–	0.403	1.0	ok
	Claude Sonnet 4.5	cap default, E9259	67	0.418	0.305	0.373	1.0	ok
	Claude Sonnet 4.5	cap default, E3223	67	0.373	0.271	0.358	1.0	ok
	Claude Sonnet 4.5	cap default, E7407	67	0.358	0.237	0.358	1.0	ok
	Claude Sonnet 4.5	closed book, cap 32,000	344	0.262	0.106	0.017	1.0	ok
	Claude Sonnet 4.5	closed book, cap default	40	0.100	0.026	0.000	1.0	ok
	Claude Sonnet 4.5	closed book, no symptom, cap 32,000	344	0.128	0.035	0.000	1.0	ok
	Claude Sonnet 4.5	neutral framing, cap 32,000, E7071	163	–	–	–	1.0	ok
	Claude Sonnet 4.5	neutral framing, cap 32,000, E8393	164	–	–	–	1.0	ok
	Claude Sonnet 4.5	neutral framing, cap 32,000, E4347	171	–	–	–	1.0	ok
	Claude Sonnet 4.5	neutral framing, cap 32,000, E8289	86	0.244	0.153	0.267	1.0	ok

continued on the next page

Table 23, continued

Arm	model	condition	n	family	mech.	site	calls	status
	Claude Sonnet 4.5	neutral framing, cap default, E8377	86	0.279	0.181	0.314	1.0	ok
	Claude Sonnet 4.5	neutral framing, cap default, E4554	40	0.325	0.132	0.300	1.0	ok
	Claude Sonnet 4.5	neutral framing, cap default, E7489	258	0.310	0.176	0.411	1.0	ok
	Claude Sonnet 4.5	with message, cap 32,000, E8972	86	0.337	0.236	0.349	1.0	ok
	Claude Sonnet 4.5	with message, cap 32,000, E1057	258	0.384	0.222	0.461	1.0	ok
	Claude Sonnet 4.5	with message, cap 32,000, E8188	67	0.373	0.271	0.388	1.0	ok
	Claude Sonnet 4.5	with message, cap default	67	0.463	0.356	0.328	1.0	ok
	Claude Sonnet 5	cap 32,000, E2085	20	0.550	0.600	0.250	1.0	ok
	Claude Sonnet 5	cap 32,000, E9126	86	0.465	0.361	0.326	1.0	ok
	Claude Sonnet 5	cap 32,000, E9295	258	0.469	0.364	0.450	1.0	ok
	Claude Sonnet 5	cap 4,000	20	0.400	0.467	0.200	1.4	1 void
	Claude Sonnet 5	cap default, E6578	3	0.667	0.333	0.333	1.3	ok
	Claude Sonnet 5	cap default, E7756	20	0.400	0.467	0.200	1.4	1 void
	Claude Sonnet 5	cap default, E4203	67	0.388	0.322	0.269	1.5	1 void
	Claude Sonnet 5	cap default, E7416	20	0.400	0.467	0.200	1.4	1 void
	Claude Sonnet 5	cap default, E2693	20	0.400	0.467	0.200	1.4	1 void
	GPT-5.6	cap 32,000	344	0.453	0.334	0.451	1.1	ok
	GPT-5.6	cap default	67	0.463	0.390	0.358	1.0	ok
	GPT-5.6	neutral framing, cap 32,000, E1390	163	–	–	–	1.0	ok
	GPT-5.6	neutral framing, cap 32,000, E6305	164	–	–	–	1.0	ok
	GPT-5.6	neutral framing, cap 32,000, E8353	171	–	–	–	1.0	ok
	GPT-5.6	neutral framing, cap 32,000, E9538	86	0.244	0.208	0.163	1.0	ok
	GPT-5.6	with message, cap 32,000	344	0.497	0.386	0.453	1.1	ok
	Claude Opus 5	cap 32,000, E2857	67	0.612	0.492	0.507	1.0	ok

continued on the next page

Table 23, continued

Arm	model	condition	n	family	mech.	site	calls	status
	Claude Opus 5	cap 32,000, E4964	86	0.523	0.417	0.442	1.0	ok
	Claude Opus 5	cap 32,000, E9786	258	0.605	0.552	0.612	1.0	ok
	Claude Opus 5	cap 4,000	20	0.600	0.667	0.350	1.6	ok
	Claude Opus 5	cap default	40	0.675	0.421	0.450	1.6	ok
	Claude Opus 5	closed book, cap 32,000	344	0.340	0.174	0.035	1.0	ok
	Claude Opus 5	closed book, no symptom, cap 32,000	344	0.198	0.080	0.020	1.0	ok
	Claude Opus 5	neutral framing, cap 32,000, E6883	163	–	–	–	1.0	ok
	Claude Opus 5	neutral framing, cap 32,000, E5647	164	–	–	–	1.0	ok
	Claude Opus 5	neutral framing, cap 32,000, E4421	171	–	–	–	1.0	ok
	Claude Opus 5	neutral framing, cap 32,000, E7177	86	0.430	0.347	0.349	1.0	ok
	Claude Opus 5	with message, cap 32,000	344	0.587	0.502	0.541	1.0	ok
	DeepSeek-R1	cap default	67	–	–	–	1.0	1 void
`flat_evidence`								
	Claude Sonnet 4.5	cap 32,000, E3454	67	0.388	0.254	0.343	6.9	ok
	Claude Sonnet 4.5	cap 32,000, E3201	67	0.418	0.271	0.313	7.0	ok
	Claude Sonnet 4.5	cap default, E7773	67	0.388	0.271	0.328	6.9	ok
	Claude Sonnet 4.5	cap default, E4742	67	0.403	0.237	0.299	7.0	ok
`graph`								
	Llama-4-Maverick	cap default	67	0.328	0.153	0.224	3.0	ok
	Claude Haiku 4.5	cap default	67	0.299	0.186	0.328	3.0	ok
	Claude Sonnet 4.5	cap 32,000, E2853	67	0.224	0.153	0.373	3.0	ok
	Claude Sonnet 4.5	cap 32,000, E9095	67	0.224	0.153	0.373	3.0	ok
	Claude Sonnet 4.5	cap 32,000, E6935	67	0.313	0.203	0.343	2.0	ok
	Claude Sonnet 4.5	cap 32,000, E6441	67	0.328	0.271	0.373	3.0	ok
	Claude Sonnet 4.5	cap 32,000, E3841	67	0.313	0.203	0.358	2.0	ok
	Claude Sonnet 4.5	cap default, E1293	40	0.300	0.132	0.375	3.0	ok
	Claude Sonnet 4.5	cap default, E3779	67	0.269	0.186	0.373	2.0	ok
	Claude Sonnet 4.5	cap default, E9731	67	0.328	0.237	0.388	3.0	ok

continued on the next page

Table 23, continued

Arm	model	condition	n	family	mech.	site	calls	status
	Claude Sonnet 4.5	cap default, E1486	67	0.373	0.254	0.358	2.0	ok
	Claude Sonnet 4.5	cap default, E1809	67	0.284	0.186	0.313	3.0	ok
	Claude Sonnet 4.5	family given 25%, cap 32,000	67	0.463	0.322	0.358	2.3	ok
	Claude Sonnet 4.5	family given 25%, cap default	67	0.478	0.271	0.313	2.3	ok
	Claude Sonnet 4.5	family given 50%, cap 32,000	67	0.642	0.407	0.373	1.9	ok
	Claude Sonnet 4.5	family given 50%, cap default	67	0.657	0.322	0.299	1.9	ok
	Claude Sonnet 4.5	family given 75%, cap 32,000	67	0.866	0.492	0.343	1.3	ok
	Claude Sonnet 4.5	family given 75%, cap default	67	0.896	0.458	0.313	1.3	ok
	Claude Sonnet 4.5	family given, cap 32,000, E3408	67	1.000	0.542	0.403	1.0	ok
	Claude Sonnet 4.5	family given, cap 32,000, E9021	67	1.000	0.559	0.358	1.0	ok
	Claude Sonnet 4.5	symptom given, cap 32,000	67	0.299	0.237	0.328	1.8	1 void
	Claude Sonnet 4.5	symptom given, cap default, E4855	67	0.343	0.237	0.343	2.5	ok
	Claude Sonnet 4.5	symptom given, cap default, E9312	67	0.418	0.254	0.373	2.1	ok
	Claude Sonnet 4.5	symptom given, cap default, E2799	67	0.388	0.254	0.343	1.8	1 void
	Claude Sonnet 5	cap default	67	0.313	0.220	0.269	3.0	1 void
	GPT-5.6	cap default	67	0.358	0.305	0.343	3.2	ok
	Claude Opus 5	cap 32,000, E9059	67	0.478	0.339	0.463	3.0	ok
	Claude Opus 5	cap 32,000, E7171	67	0.463	0.322	0.478	2.0	ok
	Claude Opus 5	family given 25%, cap 32,000	67	0.597	0.424	0.433	2.3	ok
	Claude Opus 5	family given 50%, cap 32,000	67	0.746	0.542	0.448	1.9	ok
	Claude Opus 5	family given 75%, cap 32,000	67	0.866	0.661	0.463	1.3	ok
	Claude Opus 5	family given, cap 32,000, E6946	67	1.000	0.763	0.493	1.0	ok
	Claude Opus 5	family given, cap 32,000, E6806	67	1.000	0.763	0.493	1.0	ok
	DeepSeek-R1	cap default	67	–	–	–	1.0	1 void
`graph_verify`								
	Claude Sonnet 4.5	cap default	40	0.275	0.105	0.400	7.0	ok
`graph_beam`								
	Claude Sonnet 4.5	cap 32,000	67	0.299	0.237	0.358	6.0	ok
	Claude Sonnet 4.5	cap default	67	0.343	0.254	0.313	6.0	ok
`graph_onecall`								
	Claude Sonnet 4.5	cap 32,000	67	0.373	0.203	0.373	1.0	ok

continued on the next page

Table 23, continued

Arm	model	condition	n	family	mech.	site	calls	status
	Claude Opus 5	cap 32,000	67	0.537	0.458	0.493	1.0	ok
`freeform`								
	Claude Sonnet 4.5	cap 32,000	67	–	–	0.403	1.0	1 void
`freeform_mapped`								
	Claude Sonnet 4.5	cap 32,000	67	0.313	0.220	0.403	1.0	1 void
`symptom_only`								
	Claude Sonnet 4.5	cap 32,000	67	–	–	0.418	1.0	ok
`floor`								
	top1hunks	cap default	344	–	–	0.206	0.0	ok
	top3hunks	cap default	344	–	–	0.328	0.0	ok
	majority	cap default	344	0.166	0.029	–	0.0	ok
	bow-char-lr	cap default	344	0.238	0.116	–	0.0	ok

Table 24: The three decoy strata: how each is built, how many (decoy, item) rows exist in the released pool, and how often a model claims a regression on them under neutral framing. The two count columns are in different units. *Pool* counts (decoy, item) rows; behind those rows the three strata hold 1,196, 1,084 and 1,016 distinct commits. *Scored* counts distinct commits actually evaluated, so *scored* over *pool* is not a sampling rate; the three models saw the same commits in a stratum, so one number per row, and every rate and interval in this table is on that number and not on the *pool* column. In the last row the *scored* cell counts real dev items and the *pool* cell is undefined. The candidate pools are nested by construction, each adding a constraint to the one above, but each stratum was sampled from its own pool independently, so the released strata are near-disjoint: 333 commits appear in more than one stratum, and the three distinct-commit counts sum to 3,296 against the 2,941 distinct commits in the pool as a whole. Every row carries one stratum, so the *pool* column does add to 3,642, and the three pairs of scored samples share 6, 4 and 17 commits. Comparing two rows compares two different sets of commits, not the same commits under one more constraint. Rates across the rows are not flat: every model's rate is higher on R-HOTFILE than on R-RANDOM, significant on its own for 1 of 3 models. The message filter in R-CLEAN constrains which commits become decoys; it cannot constrain what the model keys on, because this view does not show the model any commit message.

Stratum	construction	pool	scored	s4.5	GPT	OP5
R-RANDOM	same repository, same time window, sampled uniformly, absent from the corpus	1,244	171	0.468	0.175	0.158
R-HOTFILE	additionally touches a product-code file that the fix of the item it was drawn for touches	1,204	164	0.549	0.226	0.268
R-CLEAN	additionally: no repair or performance vocabulary in the message	1,194	163	0.521	0.129	0.239
real items, same framing	claims a regression	–	86	0.802	0.430	0.686

G.2 DECOY NEGATIVES AND THE THREE STRATA

Decoy construction and supported comparisons. The strata are nested as populations and near-disjoint as samples; Table 24 gives how each is built and how often each model claims a regression on it. R-RANDOM is a commit from the same repository in the same time window, sampled uniformly and absent from the corpus. R-HOTFILE additionally touches a product-code file that the fix of the item it was drawn for touches. R-CLEAN additionally requires that the commit's message contain no repair or performance vocabulary. All three strata exclude commits that were later reverted, so being unreverted is not what separates R-CLEAN from the other two. The candidate pools therefore satisfy

R-CLEAN ⊆ R-HOTFILE ⊆ R-RANDOM, but each stratum was drawn from its own pool independently, and the scored samples share only 6, 4 and 17 commits. Both contrasts below are therefore unpaired comparisons of about 163 commits against about 171 other commits, and the paired test the nesting suggests does not exist: of the 81 commit-model readings that a shared commit produces twice, 80 agree, so the overlap contains no discordant pair to test.

The message filter in R-CLEAN is a constraint on the decoys, not on the model: this view does not show the model any commit message, so the filter cannot change what the model keys on. What it does is remove commits that were plausibly performance work themselves, which is the closest we get to certifying a decoy as a true negative. The two contrasts therefore address different questions. R-HOTFILE against R-RANDOM asks whether file identity carries the decision, and the answer is weaker than the point estimates on their own suggest. Every model's false-positive rate rises when the decoy touches a file a real fix touched: Opus 5 by 11.04 points [2.27, 19.69], Sonnet 4.5 by 8.09 [−2.59, 18.52] and GPT-5.6 by 5.02 [−3.57, 13.59] (Newcombe intervals for a difference of independent proportions). The rise reaches $p < 0.05$ on its own for 1 of 3 models ($p = 0.016$ for Opus 5, 0.16 for Sonnet 4.5, 0.28 for GPT-5.6; unpaired Fisher exact), and none of them clears its Benjamini–Hochberg threshold in the whole-paper family: the smallest, Opus 5, needs 0.012.

Three models cannot push a sign test below 0.25, and an item-level permutation test that moves a commit's three model readings together (the three models scored the same commits, so their p-values are not independent) gives $p = 0.024$, which does not clear its threshold of 0.013 either.

Two confounders prevent attributing this pattern to file identity. R-HOTFILE commits are larger, median 5 product-code paths against 2 in R-RANDOM, and matching the two samples on that leaves a 6.37-point lift at $p = 0.11$. And R-RANDOM is not a control for the property as stated: 68% of its commits also touch a file some real fix touched, and only 6% touch the fix file of the item they were drawn for, which is the constraint that actually separates the two strata. What this contrast supports is that the rates are not flat across strata, not that file identity has been isolated.

R-CLEAN against R-HOTFILE asks whether the rate is inflated by decoys that are secretly real regressions, and the observed direction is consistent with contamination: every model's rate *falls* when the likely-contaminated commits are removed, and we report these the same way as the ones above: none of them clears its Benjamini–Hochberg threshold either, from 0.549 to 0.521 for Sonnet 4.5 ($p = 0.66$), from 0.226 to 0.129 for GPT-5.6 ($p = 0.029$), and from 0.268 to 0.239 for Opus 5 ($p = 0.61$; unpaired Fisher exact, the two scored samples sharing 17 commits). A fall in that direction is what the contamination account predicts: a decoy that does carry an unnoticed regression is scored as a false positive whenever a model flags it, so dropping the commits most likely to be in that state lowers the measured rate. The contrast therefore does not rule contamination out, and we read all three rates as upper bounds.

An earlier version of this paper reported the rates as flat across strata and concluded the models key on neither file identity nor message vocabulary. With one model that reading was available; with 3 it is not: the hot-file lift is positive for all 3 models, though only 1 of them reaches $p < 0.05$ alone. Whether the rate ranks the models the way recall does is stratum-dependent for the same reason; Table 21 puts each model's real-item detection rate beside its three decoy rates, which is the only frame in which the two sides of that comparison come from one model.

The rank correlation is $\rho = 1.00$ on R-CLEAN and 1.00 on R-HOTFILE, but 0.50 on R-RANDOM, where Opus 5 flags fewer decoys than GPT-5.6 (0.158 against 0.175) while recalling more real items. The coupling we report is an R-CLEAN statement over 3 models, one ordering in six by chance, and not a statement about all 9 cells.

The decoys are unvalidated negatives. We did not profile these commits. We know only that they are absent from the corpus, were never reverted, and do not describe themselves as performance work. We cannot rule out that some of them contain an unnoticed regression nobody filed, This uncertainty affects the interpretation of false positives: any such decoy is scored as a false positive whenever a model flags it, so each measured rate is an *upper* bound on the true false-positive rate. R-CLEAN is the stratum where contamination is least likely, and every model's rate on it is below that model's rate on R-HOTFILE, whose candidate pool contains R-CLEAN's. The ordering across the two is what contamination inflating R-HOTFILE would produce, so it cannot be read as evidence against contamination, and the upper bound is the only reading we claim.

The decoy pool is larger than the item pool. We built 3,642 (decoy, item) decoy negatives over 2,941 distinct commits, drawn for the 622 candidate pairs the corpus started from, of which 430 became released items; 2,157 of the commits belong to one of those. We scored 163 to 171 per stratum, which is every decoy drawn for a dev item, 471 distinct commits in all. The pool is deliberately over-provisioned: a false-positive rate is only as tight as its sample, and a reader who wants a narrower interval than ours can score more of the released pool without constructing anything, counting distinct commits and not rows, because 701 of the rows repeat a commit that another row already uses.

G.3 Prompt variants and dev selection

Decision-graph configurations. A variant counts as a graph configuration when the `arm` recorded with its run walks the graph, the field §4 counts, and not when its name looks like one: `S00-flat-baseline` and `S10-flat-vote3` are flat runs whose ids begin with `S`, and `S01-graph-baseline` is a graph run whose id contains the word. The 6 are the graph at its default settings; the walk starting at the family layer instead of the symptom layer; both extras, exclusion criteria and known fix patterns, attached to each node; the family start with fix patterns alone; beam width 2 instead of greedy descent (Yao et al., 2023); and the symptom layer supplied as an oracle. The remaining 14 vary the flat call: the prompt variants of the next paragraph, the commit-message variant of the one after, and the knobs whose ids put them in the `S` series while their runs are flat, namely the label scope at family level, exclusion criteria alone, known fix patterns alone, and full node definitions. A verify-then-descend variant that demands a discriminating prediction before committing to a branch (Weng et al., 2022) ran on a smaller item subset than the sweep, so it is not one of the 20; it is the `graph_verify` row of Table 23. Table 25 has every cell with its paired test. The parse gate is not what keeps this block negative, and the closest cell is not the voided one. `S3-extras-excl`, which attaches exclusion criteria to each label of the flat menu, ties the baseline on cause family and on fix site and stands above it on exact mechanism; its verdict is *lose* because a delta of zero on cause family is not the positive delta the frozen clause requires, and its paired test on that metric, one of the 20 in Table 25, does not separate it from the baseline either. The cell the parse gate did void, `S8-symptom-oracle`, is below the baseline on cause family and on fix site, so the void is not covering a cell that beat the baseline on the metric the rule selects on.

The exact-mechanism column is where that summary stops holding. Several rows of the `S` series print a higher exact-mechanism value than the baseline does, with `S3-extras-excl` the highest of them, and the voided `S8-symptom-oracle` is among them too. The ledger carries a paired test on exact mechanism for each of those, in `results/variants_audit.json`, and it separates none of them from the baseline; the frozen clause was written on cause family, so none of them supports a claim that the graph helps. What they do establish is the scope of the negative result: §5 concludes against the graph on cause family, the metric the selection rule and the main tables use, and not on every column of this table.

Comparison of all prompt variants. With the 6 graph configurations above, the four graph-derived knobs listed with them and the with-message arm of the paragraph below, these account for all 20 alternatives. Chain-of-thought that first states what additional work the commit causes and on which execution path (Wei et al., 2022); three-shot and six-shot demonstrations drawn from train (Brown et al., 2020); both chain-of-thought and few-shot; repository read access at the inducing commit's parent (Yao et al., 2022); repository access plus chain-of-thought; full rather than abbreviated label definitions; full definitions plus chain-of-thought; and three-sample self-consistency voting (Wang et al., 2022). The techniques in that list are published ones and each is cited, so that a reader can judge whether a loss belongs to the method or to our rendering of it on this task; the exception is the label-definition knob, which is a property of our prompt and has no outside source. Every cell, with its paired test against the flat baseline, is in Table 25. Two entries there deserve their labels read carefully. Repository access plus chain-of-thought carries the verdict *lose (site)*, which names the localisation guard, and it fails the accuracy clause too: its cause family is below the baseline and its paired test does not clear the threshold. The verdict names the stricter of two rejections, not the only one.

The self-consistency cell was misread twice and we describe the re-audit below rather than the original reading: the sampler did receive the vote count, and the *invalid* verdict the search log carried

was a false positive of our own fingerprint guard. Rebuilt from `runs.jsonl` the row is a plain *lose*. One further row needs its construction attached: `S2-scope-families` offers families rather than mechanisms, so exact mechanism is undefined for it by construction and its mech column is a dash rather than a loss.

Commit-message selection and confirmation. Supplying the commit message raised dev family accuracy from 0.418 to 0.463 over 67 items, and dev mechanism accuracy from 0.237 to 0.356 over the 59 of them whose gold label is a leaf. Those are the same four cells Table 25 prints for `B6-withmessage` and the baseline. We checked it for leakage the obvious way, splitting items by whether the message contains performance or repair vocabulary, and on 67 items the gain was *larger* on messages without those words, which we took as evidence it was not a giveaway. **That check does not survive the larger sample, and it reverses on the model that matters.** Table 27 redoes it on 344 items: GPT-5.6, the one model with a real gain, gains +16.33 points on the 49 items whose message contains such words ($p = 0.021$) and +3.05 points on the 262 that do not, where it is not significant ($p = 0.24$). Two subsets differing in their own significance is not an interaction, so we tested the difference directly by permuting the vocabulary label: 13.27 points at $p = 0.038$, which does not clear its Benjamini–Hochberg threshold of 0.016. The gain is therefore consistent with being concentrated where the message can give the answer away, and is not established as such. **Outside the pool the variant was selected on, the effect changes sign by vendor.** The dev estimate was +11.86 points over the 59 leaf-gold items of the 67-item pool the variant was picked from. On the 252 leaf-gold items outside that pool, 4.271× the 59 leaf-gold items it was selected on, GPT-5.6 keeps +5.16 points ($p = 0.053$) while both Claude models go negative (−2.78 and −1.19). Selection on a small pool is therefore doing part of the work the dev estimate appeared to report. The two denominators have to match: 277 items lie outside the pool and 252 of them are leaf-gold, so 252 against the whole 67-item pool is 3.761× and against its 59 leaf-gold items 4.271×; the second is the comparable one. We report the 67-item check and its reversal together because the reversal is the more useful result: a leak check at 67 items told us the opposite of what the same check says at 344 items. Then we ran it on 344 items with the output cap matched between the arms, and the effect changed shape (Table 28): it reversed sign on the model it was measured on (−2.25 points of exact mechanism, $p = 0.39$), stayed negative on Opus 5 (−1.93, $p = 0.38$), and was positive only on GPT-5.6 (+5.14, $p = 0.026$, nominal and short of its Benjamini–Hochberg threshold of 0.013). The split result settles three questions at once. The commit message is a real input for at least one model, so it belongs in the benchmark as a documented option with both columns reported, not as a prompt trick and not as a debunked one. A +11.86-point dev estimate turning into −2.25 points at five times the sample size, on the same model, is a sample-size lesson and not a model-ranking one. And no result on this task should be believed at 67 items. An earlier version of this paragraph added that 67 is the size at which most of the scaffolding literature we drew from operates. We withdraw that clause: we never audited the sample sizes those papers report, and the sentence named none of them, so a reader had nothing to check it against. The works our variants implement are now cited where each variant is described (Wei et al., 2022; Brown et al., 2020; Wang et al., 2022; Yao et al., 2022; Zhou et al., 2022; Khot et al., 2022; Weng et al., 2022; Yao et al., 2023), and what our 67 items support is what the paired tests of Table 25 say and nothing beyond it.

Definitions of prompt variants. A codename is not a description, and nothing above says what `S3-extras-excl` or `S7-beam2` does, so a reader cannot tell whether this sweep covered the scaffold they would have reached for. Table 26 states the knob change for every row, read off the `config` field of that run's own live row in `runs.jsonl` and not off its codename, so a knob that is renamed or rewired shows up as the knob it now is. The baseline both searches share is `arm=flat`, `view=no_message`, `labels=short`, `cot=false`, `fewshot=0`, `scope=leaves`, `extras=none`, `start_layer=N0`, `beam=`*unset*, `symptom_oracle=false`, `framing=given`, `max_tokens=`*unset*, *samples per item*=1, and `B0-baseline`, `S00-flat-baseline` are the rows that match it in every field. `arm` is the prompt shape: `flat` offers all 32 leaves in one call, `graph` walks the three layers, `flat_evidence` adds repository read tools, and `graph_beam` keeps more than one branch at the first layer; `view` is what the model sees of the commit, and `full` adds the commit message to the diff; `labels` is how much of each label's definition the prompt prints; `cot` makes the model first write what extra work the change causes and on which execution path, then choose; `fewshot` is how many solved train items are shown as demonstrations (dev never enters a prompt); `scope` chooses whether the flat menu offers the 32 leaves or the 11 families; `extras` is the extra per-leaf graph content, fix patterns or exclusions or both; `start_layer` is where the walk begins, and `N1` skips the symptom

Table 25: Every variant from both overnight searches, with the item-paired comparison against the flat baseline on cause family: n_{01} is the number of items B gets right and A does not, n_{10} the reverse, and p is the exact two-sided McNemar test on those discordant pairs. The same test on exact mechanism is in `results/variants_audit.json` for every row. Verdicts: *base* is the baseline itself, *lose* means the paired test did not clear the pre-registered threshold, *lose (site)* means it was rejected by the localisation guard, and *void* means it failed the parse/finish gate. An earlier version of this table also carried an *invalid* verdict, set when a run's prompt fingerprint equalled the baseline's; that guard misfires on knobs that change only the number of samples drawn, and no row carries it now. Every row here has a paired test, and *lose* covers either way of failing the frozen clause: a cause-family delta of zero or below is not the positive delta the first clause asks for, and a positive delta still has to clear the threshold. Where a variant was run twice, once before and once after a knob-plumbing fix, the later run is shown. Both searches ran the flat baseline (`B0-baseline`, `S00-flat-baseline`) and the two cells agree on all three metrics, so the table shows one baseline row. `S9-flat-full-defs` and `B7-fulldefs` agree the same way, on all three metrics and on their paired test: one knob, reached once from each search. Both cells were run and both are rows here. The 20 variant rows cover 19 distinct configurations, and with the *baseline* row the table prints 21 rows in all. Every cell here is rebuilt from `runs.jsonl` by `code/variants_from_runs.py`, on the live row for each variant id, so this table obeys the same append-only, supersede and void bookkeeping as every other table and uses the same denominators: *family* and *site* are over all 67 dev items and *mech.* over the 59 whose gold label is a leaf. An earlier version was generated from the two overnight search logs instead, which divided mechanism accuracy by 67, read the paired test under a field name only the second search wrote, and so printed 20 tests as 9; Appendix G records what that cost. The wrong denominator moved the *mech.* cell of 22 of the 22 variant ids in that ledger and not only the with-message row, and `results/variants_audit.json` carries both values for every entry. Every row is Claude Sonnet 4.5 at 67 items: the sweep was never run on another model, so this table bounds what prompting buys one model and carries no claim about the other five. Row bookkeeping and the re-draw band are in the paragraph headed "Row bookkeeping for Table 25".

		first draw				declared cap	
Variant	family	mech.	site	discordant	p	p	verdict
B6-withmessage	0.463	0.356	0.328	13	0.58	0.27	lose
S10-flat-vote3	0.418	0.203	0.373	4	>0.99	0.69	lose
S3-extras-excl	0.418	0.305	0.373	12	>0.99	0.55	lose
B2-fewshot3	0.403	0.237	0.343	15	>0.99	0.51	lose
B5-readrepo-cot	0.403	0.237	0.298	13	>0.99	0.77	lose (site)
B8-fulldefs-cot	0.403	0.271	0.373	13	>0.99	0.75	lose
B3-cot-fewshot3	0.388	0.203	0.328	10	0.75	0.33	lose
B4-readrepo	0.388	0.271	0.328	14	0.79	0.48	lose
S8-symptom-oracle	0.388	0.254	0.343	22	0.83	0.041	*void*
B1-cot	0.373	0.237	0.358	11	0.55	0.73	lose
B9-fewshot6	0.373	0.271	0.343	15	0.61	0.27	lose
S4-extras-fixpat	0.373	0.271	0.358	15	0.61	0.23	lose
S6-startN1-fixpat	0.373	0.254	0.358	15	0.61	0.078	lose
B7-fulldefs	0.358	0.237	0.358	14	0.42	0.23	lose
S9-flat-full-defs	0.358	0.237	0.358	14	0.42	0.23	lose
S7-beam2	0.343	0.254	0.313	13	0.27	0.031	lose
S5-graph-both-extras	0.328	0.237	0.388	22	0.29	0.15	lose
S2-scope-families	0.313	–	0.403	17	0.14	0.12	lose
S01-graph-baseline	0.284	0.186	0.313	25	0.11	0.0015	lose
S1-startN1	0.269	0.186	0.373	12	0.0063	0.049	lose
baseline	0.418	0.237	0.373	–	–	–	–

layer's irreversible commitment; `beam` is how many branches the beam arm keeps at the first layer; `symptom_oracle` hands the corpus symptom over as the first-layer answer, an upper bound and not a method; *samples per item* is majority voting over repeated calls of one unchanged prompt.

Table 26: The 20 variant codenames of Table 25 and Figure 4(a), and which knob each one moves relative to the shared flat baseline. Alphabetical by codename. A knob shown as *unset* was not passed on the command line and took the harness default; the output cap every row ran at is in Table 22. Generated by `code/make_tables.py`; the knobs themselves are explained in the paragraph above.

Variant	Differs from the baseline in
B1-cot	`cot`: `false` → `true`
B2-fewshot3	`fewshot`: `0` → `3`
B3-cot-fewshot3	`cot`: `false` → `true`; `fewshot`: `0` → `3`
B4-readrepo	`arm`: `flat` → `flat_evidence`
B5-readrepo-cot	`arm`: `flat` → `flat_evidence`; `cot`: `false` → `true`
B6-withmessage	`view`: `no_message` → `full`
B7-fulldefs	`labels`: `short` → `full`
B8-fulldefs-cot	`labels`: `short` → `full`; `cot`: `false` → `true`
B9-fewshot6	`fewshot`: `0` → `6`
S01-graph-baseline	`arm`: `flat` → `graph`
S1-startN1	`arm`: `flat` → `graph`; `start_layer`: `N0` → `N1`
S10-flat-vote3	*samples per item*: `1` → `3`
S2-scope-families	`scope`: `leaves` → `families`
S3-extras-excl	`extras`: `none` → `exclusions`
S4-extras-fixpat	`extras`: `none` → `fix_patterns`
S5-graph-both-extras	`arm`: `flat` → `graph`; `extras`: `none` → `both`
S6-startN1-fixpat	`arm`: `flat` → `graph`; `extras`: `none` → `fix_patterns`; `start_layer`: `N0` → `N1`
S7-beam2	`arm`: `flat` → `graph_beam`; `beam`: *unset* → `2`
S8-symptom-oracle	`arm`: `flat` → `graph`; `symptom_oracle`: `false` → `true`
S9-flat-full-defs	`labels`: `short` → `full`

Variant records and variation across draws. The ledger holds one id more than the 21 rows there: both overnight searches contributed a flat baseline, the two score identically on all three metrics, and they print as the single *baseline* row. `S2-scope-families` offered cause families rather than leaf mechanisms, so its *mech.* cell is a dash and not a measured zero. `S10-flat-vote3` is the one cell whose per-item file is not one row per item: it writes 201 rows for 67 items, one per vote, so it is excluded from the re-draw comparison. Re-drawing a configuration moves cause family by 0 to 6 items, median 1, over the 19 configurations that carry two draws, and the band is the largest of those. The two arms that also carry a paired McNemar move 2 and 4 items, and an earlier version of this paragraph took the band from those two alone: that gives 4 rather than 6, and the narrower band is the self-serving one, because everything outside the band is below the flat call, so a band calibrated on two configurations moves cells out of "unresolved" and into "worse than one flat call". The re-draw is not the same configuration twice: the second draw also raised the requested output cap from 4,096 to 32,000, so this band mixes sampling noise with whatever changing the cap does, and it is a noise band only under the separate finding that 4,096 never bound these calls. Those two moves are symmetric, which is what makes them readable as noise rather than as a cap effect: 10 items change answer between the flat draws, split 4 to 6 ($p = 0.75$ against a threshold of 0.042), and 12 change between the graph draws, split 8 to 4 ($p = 0.39$ against a threshold of 0.033). What moves between the draws is the asymmetry and not the amount of disagreement, and both of these counts mix the cap change in with the sampling. 16 of the 19 variants that can be placed on that scale are within that band of the flat baseline, so most of the ordering in that table is not resolved at 67 items, and the 3 that lie further out are all *below* the baseline. Every number in the table is the first draw, at 4,096 tokens rather than the 32,000 Setup declares; the response cache shows 4,096 did not bind them, and all 22 of these cells, counting both baseline draws, were re-drawn at 32,000. That later draw is the one Figure 4(a) plots as its rows, with this draw as the grey dashes beside them, and the last p column of the table is that later draw's own item-paired comparison against the flat baseline: no variants sit above the baseline on cause family there and no variants separate upward from it. 4 variants separate *downward*, `S01-graph-baseline`, `S7-beam2`, `S8-symptom-oracle`, `S1-startN1`, and the deepest of those is not a prompt at all but the graph walk itself, the comparison §5 reports; the two are one test and enter the whole-paper family once, so 18 of these 19 distinct tests

are new to it. `S8-symptom-oracle` is likewise an upper-bound arm that is handed the symptom rather than a scaffold a user could deploy. On exact mechanism the same re-draw tests 19 of the 20 cells, the undefined one being `S2-scope-families`, which offers families rather than leaves, and no variants separate from the flat call in either direction. The two variants named in §4 were picked because their point estimate is above the baseline, which is the reading that makes a null easiest to state: `S3-extras-excl` at 19/59 ($p = 0.55$ against a threshold of 0.036, so we report the direction and not a rejection) and `B8-fulldefs-cot` at 18/59 ($p = 0.75$ against a threshold of 0.042, so we report the direction and not a rejection), against the flat call's 16/59. Every remaining cell's test is in `results/sweep32k.json` under `exact_mechanism_at_declared_cap`.

Table 27: Is the one surviving gain a giveaway? The same two arms as Table 28, split by whether the inducing commit's message contains performance or repair vocabulary (the same regular expression the corpus census uses). *only* + counts items only the with-message arm gets right, *only* − the reverse, and p is the exact two-sided McNemar test on those. For GPT-5.6 the gain is significant only on the subset where the message can name the problem.

Model	message	n	−msg	+msg	Δ	only +	only −	p
Claude Sonnet 4.5	has perf words	49	0.204	0.184	−0.020	1	2	>0.99
Claude Sonnet 4.5	has none	262	0.256	0.233	−0.023	20	26	0.46
GPT-5.6	has perf words	49	0.245	0.408	+0.163	9	1	0.021
GPT-5.6	has none	262	0.351	0.382	+0.031	22	14	0.24
Claude Opus 5	has perf words	49	0.408	0.490	+0.082	4	0	0.12
Claude Opus 5	has none	262	0.542	0.504	−0.038	9	19	0.087

Table 28: Whether the commit message counts as part of the item, on 344 items with the output cap matched between the two arms. The dev-only gain does not survive on the model it was found on; it reverses. It is positive on GPT-5.6 (+5.14 points of exact mechanism, $p = 0.026$, nominal only: the whole-paper Benjamini–Hochberg threshold for it is 0.013). The Δ values quoted in §4 are computed on the items both arms scored, under the same construction-undefined exclusion as every other table, so they match the difference of the two columns here to rounding. Paired tests are in Table 33. This table is the reason the benchmark documents the message as an input *option* with two reported columns rather than picking one.

Model	fam. −msg	fam. +msg	mech. −msg	mech. +msg
Claude Sonnet 4.5	0.369	0.372	0.248	0.225
GPT-5.6	0.453	0.497	0.334	0.386
Claude Opus 5	0.584	0.587	0.521	0.502

Free-form diagnosis without the label menu. The sharpest form of "this is just a multiple-choice test" is to delete the choices. We ran Sonnet 4.5 with no taxonomy in the prompt at all, asked for a free-text mechanism description, and then had GPT-5.6, a different vendor's model shown only the free text and never the diff, place each description in the taxonomy. Result: cause family 0.313 against 0.448 with the menu, and 31% of the free-text answers cannot be placed under any label.

The mapping step is not what loses the accuracy. Fed 32-label answers that are already valid, the mapper returns all 60 of them unchanged, so on well-formed input the link adds no measurable noise. That rate (1.00) is measured on those 60 items only.

This cell does not pass the pre-registered parse gate (31% unparsed against a 0.10 allowance) and is recorded as void in `results/tables/app_allruns.tex`, among the 12 voided rows nothing has replaced. Its numbers are printed above, and the gate was not moved to accommodate them: the gate was frozen before any of these runs. The gate is also the wrong instrument for this one arm. Everywhere else, "unparsed" means the harness could not read the model's answer. Here it means a human-readable diagnosis does not correspond to anything in our taxonomy, which is a result about the coverage of the taxonomy and not a failure to read an answer. Read against the panel's "no label fits" rate of at most 0.039 on the same taxonomy, the gap says most unplaceable answers are vague rather than genuinely outside the label set.

G.4 Fix-site scores under two coordinate systems

All 4 site criteria the scorer computes, and the one Table 1 prints. `score_m2` writes 4 hit criteria per item, at file, line and symbol granularity and for the union of the last two, plus the file-level precision and recall its own docstring requires; Table 1 prints the line one and the aggregation dropped the rest. Table 29 gives all of them on the same 344 items, and each of the three we had left out reads against us somewhere. Under the union criterion GPT-5.6 leads at 0.648 where Claude Opus 5, first on cause family, reads 0.619, and the paired test on that reordering does not separate ($p = 0.33$ against a threshold of 0.031; 48 items the leader hits and Claude Opus 5 misses against 38 the other way); with 4 of the 6 models inside 0.029, narrower than the half-width of any one of their intervals, that criterion orders nothing. At symbol level Claude Opus 5 is last of 6 at 0.215 against 0.549 for Claude Sonnet 4.5, the two disagreeing on 143 items one way and 28 the other; it is the model that fills the optional `symbol` field least often, so that column reads answer format as much as localisation. We print no test for it: the comparison is post hoc, and a p enlarges the family every other test in this paper is corrected against, so the discordant counts are what we give. At file level the 6 models order exactly as cause family does, $\rho = 1.000$ with 0 inversions, against 0.829 and 2 inversions at line level, 0.486 and 5 under the union, and -0.086 and 8 at symbol level. Which granularity is read therefore decides whether pointing looks like a second ability or like a restatement of naming, and §4.1 reads the granularity whose ordering departs from cause family by the smallest non-zero margin.

Score granularity and cited-site precision. The asymmetry §4.1 rests on is an effect size within one model, and it survives all 4 criteria. Across the 27 configurations of Claude Sonnet 4.5 that Appendix H uses, cause family moves with a between-configuration standard deviation of 0.055 against 0.033 at file level, 0.039 at line level, 0.036 at symbol level and 0.028 under the union, so the asymmetry holds at every granularity: all four site criteria move across configurations by less than cause family does. Granularity moves the between-model ordering, not the spread a prompt change produces. Two further omissions are in the released scorer itself. `score_m2` computes file-level precision and recall per item and the aggregation kept neither, although its docstring says both must be reported because recall alone rewards citing the whole diff; the second half of Table 29 gives them, and precision is the direct check on a citation budget that runs from 1.26 to 2.91 files per answer against the at-most-3 instruction. `score_m4` defines a second joint metric we never named, `m4_no_disc`, exact mechanism together with the union site criterion, which is Table 1's *Both* with the looser site half: on the same 311 items it runs 0.116 to 0.395 where *Both* runs 0.080 to 0.376. *Both* is the stricter of the two and stays the one we quote; we print this one so the release cannot be read as carrying an unreported headline.

The fix-site metric compares two coordinate systems, and the direction of the error is against us. The lines the fix commit changed are recorded in the fix commit's parent, and the model cites lines it read in the inducing commit's diff. Those are different revisions, so the ±12-line window is applied to line numbers that may have shifted. We measured the size of that shift. Mapping every truth line from the fix's parent into the inducing commit's coordinates with the hunk offsets between the two revisions, 2,052 of 12,186 truth lines (17%) fall inside a region that changed in between and therefore have no counterpart at all, and 34 of the 619 corpus items with product-code truth lose every truth line, 11 of them inside the 344 items Table 1 scores. On the rest, rescoring every saved answer against the mapped lines raises pooled line-level accuracy from 0.356 to 0.496, raises the three-largest-hunks floor from 0.328 to 0.413, and leaves the point-estimate ordering of the 6 models unchanged, an ordering that, as the tests below show, this column does not resolve in any frame. Two consequences. The fix-site column we print is a *lower* bound, by about 14.06 points pooled, so nothing in the paper is inflated by this. And the tie between the weakest model and the zero-call floor survives the remapping: Llama-4-Maverick reads 0.395 against the floor's 0.412 (the pooled 0.413 above is over all 344 items, this one over the 342 that both sides score) on the 342 items both score, 74 discordant, exact McNemar $p = 0.56$, so the remapping narrows the gap and leaves the two where the printed scoring leaves them, indistinguishable. We keep the unmapped scoring as the headline because it needs no repository checkout to reproduce, and we ship the remapped rescoring (`code/rescore_m2.py`, `results/m2_recoord.json`) so the comparison can be checked.

Raw and remapped scores for each model. Table 1's fix-site column and its remapped counterpart, on the same 344 items with an unreadable answer counted as wrong in both: Llama-4-Maverick

Table 29: Every site criterion `score_m2` computes, on the 344 scored items of Table 1, one flat call per item. *file* is a cited file that the fix also changed, *line* a cited line within 12 lines of one the fix changed, *symbol* a cited `symbol` sharing an identifier with the fix hunk's enclosing context, and *line or symbol* the union the scorer calls the mechanism site. Intervals are Wilson to two decimals. The second tabular adds the two quantities the scorer's docstring requires and the aggregation dropped, file-level precision and recall averaged over items with an unparseable or site-free answer counted as zero, the mean number of distinct files cited, and `m4_no_disc`, the scorer's undocumented conjunction of exact mechanism with the union criterion, on 311 items.

Model	k	n	rate	95% CI
file				
Llama-4-Maverick	231	344	0.672	[0.62, 0.72]
Claude Haiku 4.5	250	344	0.727	[0.68, 0.77]
Claude Sonnet 4.5	265	344	0.770	[0.72, 0.81]
Claude Sonnet 5	271	344	0.788	[0.74, 0.83]
GPT-5.6	269	344	0.782	[0.74, 0.82]
Claude Opus 5	301	344	0.875	[0.84, 0.91]
line (the criterion Table 1 prints)				
Llama-4-Maverick	105	344	0.305	[0.26, 0.36]
Claude Haiku 4.5	140	344	0.407	[0.36, 0.46]
Claude Sonnet 4.5	154	344	0.448	[0.40, 0.50]
Claude Sonnet 5	144	344	0.419	[0.37, 0.47]
GPT-5.6	155	344	0.451	[0.40, 0.50]
Claude Opus 5	196	344	0.570	[0.52, 0.62]
symbol				
Llama-4-Maverick	136	344	0.395	[0.35, 0.45]
Claude Haiku 4.5	106	344	0.308	[0.26, 0.36]
Claude Sonnet 4.5	189	344	0.549	[0.50, 0.60]
Claude Sonnet 5	183	344	0.532	[0.48, 0.58]
GPT-5.6	177	344	0.515	[0.46, 0.57]
Claude Opus 5	74	344	0.215	[0.17, 0.26]
line or symbol				
Llama-4-Maverick	172	344	0.500	[0.45, 0.55]
Claude Haiku 4.5	174	344	0.506	[0.45, 0.56]
Claude Sonnet 4.5	219	344	0.637	[0.58, 0.69]
Claude Sonnet 5	215	344	0.625	[0.57, 0.67]
GPT-5.6	223	344	0.648	[0.60, 0.70]
Claude Opus 5	213	344	0.619	[0.57, 0.67]

Model	file prec.	file rec.	n cited	mech. and site
Llama-4-Maverick	0.618	0.503	1.26	0.116
Claude Haiku 4.5	0.611	0.582	2.71	0.167
Claude Sonnet 4.5	0.639	0.608	1.60	0.212
Claude Sonnet 5	0.604	0.651	2.91	0.273
GPT-5.6	0.686	0.615	1.41	0.296
Claude Opus 5	0.654	0.722	1.82	0.395

0.305 to 0.392, Claude Haiku 4.5 0.407 to 0.529, Claude Sonnet 4.5 0.448 to 0.558, Claude Sonnet 5 0.419 to 0.552, GPT-5.6 0.451 to 0.581, Claude Opus 5 0.570 to 0.701, and the three-largest-hunks heuristic 0.328 to 0.413 (top hunk only, 0.206 to 0.262), which is 14.06 points pooled over the run list `results/m2_recoord.json` freezes, a single difference taken over every run in that list and not the gap between the two columns just listed for any one model. The 5 models that separate from the heuristic in the printed frame all separate in the remapped one and no others do (5 of 6), by paired exact McNemar on the items both sides score: Claude Haiku 4.5 p <0.001, Claude Sonnet 4.5 p <0.001, Claude Sonnet 5 p <0.001, GPT-5.6 p <0.001, Claude Opus 5 p <0.001.

Llama-4-Maverick separates in neither frame ($p = 0.38$ printed, 0.56 remapped): over the 344 items only the model is right on 28 and only the heuristic on 36.

Coverage of coordinate remapping. On the scored pool, 985 of 6,804 truth lines (14%) have no counterpart in the mapped coordinates, and `code/rescore_m2.py` counts the 11 items that lose all of theirs as wrong, so the remapped column is itself a lower bound on them. Dropping those items from the denominator leaves 333 items and moves no cell by more than 2.31 points: Llama-4-Maverick 0.405, Claude Haiku 4.5 0.547, Claude Sonnet 4.5 0.577, Claude Sonnet 5 0.571, GPT-5.6 0.601, Claude Opus 5 0.724, heuristic 0.426. Llama-4-Maverick stays below the heuristic in that reading too.

Coordinate drift and localisation accuracy. On three of the four exposure measures we can compute, yes in part, and the fourth says nothing, so we report all four. Per item, against how many of the 6 models hit, on 344 items with a 10,000-resample item bootstrap: the largest net shift of the item's truth lines gives Spearman −0.387 [−0.478, −0.290], the number of commits from the inducing to the fixing commit −0.354 [−0.448, −0.256], and calendar days between them −0.347 [−0.440, −0.249]. On the fraction of truth lines with no counterpart, which is the measure the objection names first, there is no association at all: −0.036 [−0.137, 0.069]. After remapping, the line-shift association is gone (−0.032 [−0.137, 0.073]) while commit distance keeps −0.138 [−0.243, −0.031] and calendar distance keeps −0.159 [−0.262, −0.049], both intervals excluding zero. Two readings of the remapped column that do not favour it: an association with the unmappable fraction *appears* after remapping where the printed column had none (−0.116 [−0.228, −0.002]), and the gain from remapping is smallest on the items that lose the most truth lines (−0.135 [−0.255, −0.011]), that is, the remapped reading is weakest exactly where the printed one is most suspect.

Scores on items without coordinate drift. On the 96 items whose truth lines do not move at all the two scorings are identical item by item, so the gain there is 0.000, and 100% of the flips falling on the 248 items that do move is true by construction: it is a check that the two strata are defined correctly, not evidence that the confound is contained. The measurement is the contrast between the strata. As printed, the moved items score −0.281 [−0.370, −0.193] against the drift-free ones; remapping removes 0.165 [0.121, 0.210] of that gap and leaves −0.115 [−0.203, −0.027], an interval that excludes zero. So 41% of the raw gap is not coordinates, and we do not know what it is: commits with more churn between introduction and fix may simply be harder to localise, and a hunk-offset mapping may also leave mis-scoring behind. Nothing here separates those two, and only a per-item human check could.

Matched baseline on items without coordinate drift. Those 96 items give a coordinate-clean localisation reading, and they are also an easier subset: the three-largest-hunks heuristic reaches 0.469 on them against 0.328 on the full pool, so their model numbers have to be read against that floor and not against Table 1. They are Llama-4-Maverick 0.448, Claude Haiku 4.5 0.625, Claude Sonnet 4.5 0.656, Claude Sonnet 5 0.604, GPT-5.6 0.688, Claude Opus 5 0.792, that is Claude Opus 5, GPT-5.6, Claude Sonnet 4.5, Claude Haiku 4.5, Claude Sonnet 5, Llama-4-Maverick, with 5 above the floor on the point estimate. 2 models sit at a different rank than in Table 1, and that move does not separate: Claude Haiku 4.5 against Claude Sonnet 5 is $p = 0.84$.

Revision choice and changed-hunk mapping. Two free choices, not one. The target can be the inducing commit, whose diff the model is shown, or its parent, where the read-only tool view is pinned; and a truth line falling inside a hunk that changed between the two revisions can be dropped as having no counterpart, as `code/rescore_m2.py` does, or clamped to the nearest position inside the target hunk, as its own first version did. Mapping into the inducing commit and dropping is the 14.06 points above. Mapping into the parent with the clamp policy gives 9.19 points pooled, per model between 3.49 and 15.12, and an ordering that reads Claude Opus 5, Claude Sonnet 5, Claude Sonnet 4.5, GPT-5.6, Claude Haiku 4.5, Llama-4-Maverick: 2 models exchange ranks, and the exchange does not separate ($p = 0.17$; the same pair does not separate in the printed column either, $p = 0.18$). The fourth cell, the parent with the drop policy, reads −25.09 points and moves 4 models, and it is not a defensible reading of the same quantity: 162 of the 344 items keep no target line at all and 4,696 of 6,804 truth lines (69%) have no counterpart, against 0 unscoreable items under the clamp policy, so both its delta and its ordering are computed on whichever items survive.

Table 30: Line-level fix-site accuracy at 4 window half-widths, ±0, 6, 12 and 25 lines, on the 344 items of Table 1 in the printed frame, an unreadable answer counted as wrong. † marks the width the paper scores at, whose column reproduces Table 1 item by item.

Arm	±0	±6	±12†	±25
Llama-4-Maverick	0.084	0.250	0.305	0.407
Claude Haiku 4.5	0.180	0.363	0.407	0.480
Claude Sonnet 4.5	0.215	0.404	0.448	0.544
Claude Sonnet 5	0.122	0.358	0.419	0.526
GPT-5.6	0.177	0.398	0.451	0.535
Claude Opus 5	0.265	0.485	0.570	0.648
three largest hunks (heuristic)	0.044	0.244	0.328	0.436
largest hunk only (heuristic)	0.032	0.148	0.206	0.282

Our own revision ledger recorded that cell's reordering as a reason to distrust the column; its figures are not reproducible from the shipped scripts and are not printed here.

Sensitivity to line tolerance. The ±12-line window is a free parameter, it is hard-coded in the scorer, and until this paragraph no version of this paper swept it. Swept over 4 half-widths, ±0, 6, 12 and 25 lines, in the printed frame and through `probe/score.py`'s own `score_m2`, it reproduces Table 1's fix-site column item by item at ±12 and moves every arm a long way at the others (Table 30). Two readings follow. The *level* of this column is a property of the window as much as of the models: the three-largest-hunks heuristic runs from 0.044 when a cited line has to land on a changed line exactly to 0.436 at the widest window, and Claude Opus 5 from 0.265 to 0.648, so no fix-site number here is comparable with one scored under a different window. The *comparison* §4.1 draws against that heuristic changes sign inside the swept range, which is the sharper cost. Llama-4-Maverick is above the heuristic at the two narrowest widths, 0.084 against 0.044 ($p = 0.038$ against a threshold of 0.016) and 0.250 against 0.244 ($p = 0.89$ against a threshold of 0.045), and below it at the two widest ($p = 0.38$ against a threshold of 0.033 at the published width, $p = 0.28$ against a threshold of 0.03 at the widest). None of the four clears the whole-paper correction, and the three widths the paper does not score at enter the family as three new tests, so the wording §4.1 uses, a tie, is the one every width supports and "below the floor" is supported by none of them. The other 5 models stay above the heuristic on the point estimate at every width. The file-level and symbol-level parts of the fix-site score do not use the window and are unchanged; every reading built on the line-level hit, including Table 1's Both column, moves with it. `code/line_tol_sweep.py` writes `results/line_tol_sweep.json` and refuses to write it if the published width stops reproducing the printed column item by item.

Uncertainty in model rankings. 2 of the 5 adjacent-rank pairs in Table 1's fix-site column separate at 0.05 by paired exact McNemar and the rest do not, so the column ranks groups rather than 6 models. Of the 10 paired tests on the pairs that sit in a different order under one of the frames or subsets above, 0 reach 0.05. Every rank statement in this appendix, including the ones we raise against ourselves, is a point estimate.

One estimator caveat. `code/items.py` and `code/rescore_m2.py` do not apply the same product-code file filter, so the truth lines the paper scores against and the truth lines the remapping starts from differ on 2 of the model-item cells, on 1 item. Every per-model pair above compares the printed column with the remapped one directly, so each of them straddles the two filters and inherits that gap; the pooled 14.06 points is a difference taken inside the remapping pipeline, under one filter on both sides, and does not.

Presentation differences in the family-oracle comparison. The multiple §5 prints compares two arms that differ in more than the family label. Rendering a layer of the graph emits, for every candidate, the edge's branch question, what taking that branch means, and the sibling exclusions; the flat menu emits none of that, only the 32 leaf definitions. The oracle arm goes through the layer renderer, so it receives that text and the flat arm does not. The multiple therefore mixes three

ingredients: the family label itself, the shortened candidate menu, and the per-edge annotations. We have not separated them, and the direction is against us on its face, because the arm that wins is the one receiving the extra text. So we measured it rather than bounding it from other cells: the oracle arm was re-run on the same items, at the same cap, with the same view and framing, with that text and nothing else removed. neither multiple falls once that text is removed. Sonnet 4.5 goes from 2.00× to 2.06× (33 of 59 against 16 for the flat arm) and Opus 5 from 1.55× to 1.55× with the same numerator; the two oracle arms disagree on 13 and 10 items and neither difference separates (p >0.99, p >0.99). The prediction we set out to test, that the corrected multiple would fall materially, is not what the data show, and we report that the same way we would have reported the fall. §5 prints the matched arm, by a rule fixed before the re-run. What the multiple still mixes is the family label with the shortened menu, which no arm in this design separates.

G.5 Zero-call baselines and trained routers

Baselines for the memorisation control. The floor printed everywhere is a cross-repository floor: folds hold out a whole repository, so a fold trains on as few as 147 items over 32 classes. The regime a user actually faces, diagnosing a repository that has its own history, is a within-repository floor, and that one is *higher*: 0.256 on cause family against the same arm's cross-repository 0.238 (both are the char-n-gram arm; the floor Table 1 prints for that column is the encoder arm's 0.253, which is not what this fold comparison is about). The gap is smaller than the spread across fold draws and a paired test does not separate the two, so the printed number stands; `code/floor_within.py` reports six fold regimes and every model against each. One cell does not survive the comparison, and it is the one the paper leans on least safely: Sonnet 4.5's *Diff withheld* score, 0.262, is not separated from the cross-repository floor at all (p = 0.55 paired), and against the within-repository floor it is above in only 8 of 10 fold draws. So the memorisation control for that model is not merely close to the floor, it is inside it, which is the strongest form of the warning §6 gives. Against the floor that matches what that arm is actually shown, neither cell clears it. The *Diff withheld* prompt still names the observed symptom class, and a zero-call lookup table from that class to its training-fold modal cause family reaches 0.331 on the 344 scored items under the same folds (`code/symptom_prior.py`), so the stronger cell 0.340 is not separated from it either (p = 0.86). Neither verdict turns on the tie-breaking rule, although the estimate does: 3 fold-by-symptom cells have a tied modal family, and over every way of breaking those ties the floor runs across [0.299, 0.349] while both verdicts stand. The symptom is a weak constraint by construction: it admits 7.19 of the 11 cause families on average, and on 19 of the 295 items with a named symptom (0.064) the gold family is not reachable from the annotated symptom along the graph's own edges, an inconsistency between our answer key and our taxonomy, and a ceiling of 0.936 on any router constrained to a symptom's admissible families. Read as the printed floor is read, leaf first and family off the leaf, the same table gives 0.276 or 0.294 depending on that same `n/a` treatment, both above the 0.238 char-n-gram floor that reads the diff.

A supervised router. The routing bottleneck of §5 invites the response that nobody trained a router. We trained 6, all char-n-gram classifiers over the inducing diff with repository-held-out folds. Two of them are flat, and those two are not the same number. The char-n-gram floor of §2, 0.238, is a *leaf* classifier whose family prediction is read off its leaf output, which is how that row is produced everywhere else in the paper. A classifier trained on the 11 families directly reaches 0.253 on train and dev and 0.299 on the selection pool. Both are above the 0.166 majority-class rate under the same folds and above the 0.224 that Sonnet 4.5's own graph walk achieves, and both are below both models' flat family readings (0.448 and 0.612) and below the 0.460 break-even. Two denominators for that graph-walk cell are already on the page at once: 0.224 here divides by 67 items, counting the item the model never answered as wrong, while the matching row of Table 22 prints the run record's own 0.227, which drops that item and divides by 66. We report the inconsistency rather than quietly reconcile it; every number in this paper recomputed from per-item files rather than copied from a run record uses the 67-item convention. Those readings are all on the 67-item routing axis; on the 59 leaf-gold items §5 matches the denominators to, the same classifier reads 0.305 against thresholds of 0.469 and 0.664, and the comparison does not turn on which axis or which crossing estimator we use. So the answer to that response is not that nobody trained a router. The classifier we trained is worse than asking the model for a family and reading the family off its answer, and that reading is the baseline any supervised router has to beat here.

Learned zero-call baselines. The obvious objection to a char-*n*-gram floor is that a modern pretrained code representation would be a much stronger comparator, which is a measurable claim. `code/baseline_embed.py` measures it. It freezes microsoft/unixcoder-base (126 M parameters, revision `5604afdc964f`), embeds each inducing diff in 1,019-token windows, pools the windows two ways (mean, and mean concatenated with max), and fits the identical multinomial logistic head on the identical leave-one-repository-out folds, scored by the identical code, as the char-*n*-gram floor: the representation is the only thing that changes. A third arm replaces the representation with the changed file paths, their directory prefixes, the diff size and the repository name, which separates "a pretrained representation" from "any feature stronger than character *n*-grams". Our pre-registration for this measurement fixed the encoder, the two window poolings, the head and the folds before the run, and fixed what would be printed: 3 arms are pre-registered, the hunk ranker below is not. It also discloses that the two poolings had already been compared once in a scouting pass that wrote no artifact, so the pooling choice is exploratory rather than pre-registered; both are printed and neither is dropped.

The encoder is fed in the format its own documentation prescribes. An earlier version of this measurement built each window as `[<s>]` + content + `[</s>]`, never emitted the `<encoder-only>` mode-control token that UniXcoder's reference implementation prescribes for encoder-only embeddings, and averaged content tokens only rather than all non-pad positions as that implementation's own sentence embedding does. Both knobs are measured here. Across the 8 resulting cells (Table 32) cause-family accuracy spans 0.038: the undocumented invocation reads 0.265 and the documented one 0.253. Table 1 and every other print site use the documented cell, because that is the cell a reader who follows the model card reproduces; the arm is weaker than the undocumented number made it look, and an earlier draft of this appendix printed the larger one. That rule, that only an invocation following the model card is eligible to be the printed floor, was fixed before the corrected run, and both values are printed either way. Note that 0.038 of spread across four ways of feeding one frozen encoder is larger than the 0.015 this arm gains over the char-*n*-gram classifier. An arm with that much give is a floor, not a measurement of what pretrained representations can do.

What it changes. Criterion 1 of the pre-registration prints, in Table 1 and in the Floors paragraph of §2, the strongest zero-call arm in each column, whichever arm that is. The cause floor therefore moves from the char-*n*-gram 0.238 to 0.253 (UniXcoder, mean), a gain of 0.015 that the paired exact McNemar test does not separate ($p = 0.66$): the encoder is right where the char classifier is wrong on 43 items and wrong where it is right on 38. Both window poolings reach the same cause-family count, 0.253 and 0.253; ties are broken by arm name, so the printed cell is the mean-pooled one. Exact mechanism goes the other way: the encoder reads 0.084 and 0.096 against the char classifier's 0.116, so the char arm keeps that floor and the difference is again not separated ($p = 0.18$ and $p = 0.46$). The structural arm is below the char classifier on both columns (0.195 cause, 0.035 mechanism), and on mechanism that loss is the one comparison in this group that does separate ($p < 0.001$). The answer is therefore per column and with a p attached: on cause family this pretrained encoder is above the char-*n*-gram floor and not separated from it, on exact mechanism it is below and not separated, and nothing here supports "much stronger" in either direction. It does not support the reverse slogan either: we do not claim that a pretrained representation cannot beat character *n*-grams, only that this one, frozen, with a linear head, on these 344 items, does not.

Raising the printed floor costs a separation. All 6 models stay above the cause floor by point estimate, but "above" and "separated from" are different claims, and the higher the printed floor the weaker the second one gets. Against the printed 0.253 the six paired tests read $p < 0.001$ (Opus 5), $p < 0.001$, $p < 0.001$, $p < 0.001$, $p = 0.012$ (Haiku 4.5, which does not clear the whole-paper correction of Appendix H at rank 39 of 186, threshold 0.01) and $p = 0.057$ (Llama-4-Maverick, which does not clear it at rank 69, threshold 0.019): so we report the direction and not a rejection for the weakest model. Claim **(C1)** of §1 is a point-estimate claim at this floor for that model. Every model's margin over the floor shrank by exactly the 0.015 the floor rose, which is what costs the weakest one its rejection; criterion 1 fixed that trade before the numbers existed.

The printed floor is a maximum. It is the largest of 9 zero-call arms (21 counting the undocumented cells of Table 32), taken with no correction for that selection, so as an estimate of any single arm's accuracy it is biased upward; the 0.015 encoder gain is itself a maximum over the 3 pre-registered arms minus a fixed comparator. We print it anyway, because criterion 1 fixed the rule before the measurement and the alternative is to pick an arm after seeing the numbers. Table 31 gives every arm, so a reader who wants a single-arm floor can take one.

An exploratory fix-site arm. The same encoder can rank hunks: score every hunk of the inducing diff with the same head on the same folds and cite the top k. That reaches 0.198 at $k = 1$ and 0.294 at $k = 3$, against 0.206 and 0.328 for citing the largest and the three largest hunks ($p = 0.77$ and $p = 0.11$), so it does not become the printed fix-site floor. Two things keep it exploratory. It is not in the pre-registration. And its positive labels are defined by the metric it is scored against: a hunk is positive when its own (file, line) already falls inside the ±12-line window of §2, which is also how the heuristic floor is scored, and although the ranker never sees a fix commit and never sees its own repository during training, that circularity has to be stated rather than argued away. 0 of the six models fall below it by point estimate, and 1 is not separated from it: Llama-4-Maverick reads 0.305 against 0.294 ($p = 0.70$), the same model that ties the heuristic floor in §4.1. Under the undocumented invocation this arm reads 0.299 instead, the same sensitivity as above.

Cost, hardware, denominators. The encoder pass is 8,761 forward windows over 344 diffs (median 5 windows each) plus 6,362 hunks, of which 515 are positive: at most 15 wall-clock minutes and 7 CPU-hours on 32 CPU threads, and no GPU. Those two figures are upper bounds on one host's measurement rather than the measurement itself, because a wall-clock number is the one quantity in this pipeline that changes between runs, so we print a bound instead of a macro that moves. CPU-only is a choice and not a hardware limit: this host's NVML is broken, so `nvidia-smi` fails, but a GPU is otherwise reachable, and we keep the floor CPU-only so that it can be re-run without one. We did not try a larger cached embedding model, so this is one 126 M-parameter code encoder and not the strongest pretrained representation available; the claim is about this comparator. Every arm here shares Table 1's items and denominators, 344 for cause family and fix site and 311 for exact mechanism, with an unreadable answer counted wrong and kept in the denominator, and the script refuses to write its output unless all six models' three printed cells reproduce from the run files cell by cell.

Table 31: Every zero-call baseline this paper measures, on the 344 scored items with the folds and the scorer of §2. Criterion 1 of the pre-registration prints the strongest cell of each column in Table 1: UniXcoder, mean for cause family, char n-gram TF-IDF for exact mechanism, the three largest hunks for the fix site. A dash means the arm does not produce that column. The two hunk-ranker rows are exploratory and not pre-registered. `code/make_tables.py` writes this table from `results/baseline_embed.json`.

Zero-call baseline	representation	Cause	Mechanism	Fix site
majority class	–	0.166	0.029	–
char n-gram TF-IDF	char 3–5gram	0.238	0.116	–
UniXcoder, mean over windows	pretrained	0.253	0.084	–
UniXcoder, mean ‖ max	pretrained	0.253	0.096	–
changed paths, dirs, diff size, repo	structural	0.195	0.035	–
one largest hunk	none, no fitting	–	–	0.206
3 largest hunks	none, no fitting	–	–	0.328
UniXcoder hunk ranker, $k = 1$	pretrained	–	–	0.198
UniXcoder hunk ranker, $k = 3$	pretrained	–	–	0.294

Single-hunk localisation baseline. The three-largest-hunks heuristic reaches 0.328 at line level; citing only the single largest hunk already reaches 0.206, so most of that floor is one hunk and the rest of the heuristic adds little.

Hierarchical routing and inference. §5 reports the whole deficit against the flat menu and says that none of its ingredients separates, routing and shape alike, so the shape claim needs a trained control and not only a prompted one. `code/hier_router.py` trains and scores all 6 of them, plus one upper-bound arm handed the correct family, on the identical repository-grouped folds, the identical char-n-gram encoder and the identical scorer as the floor of §2, and reproduces that floor exactly, so flat against hierarchical is a controlled contrast. Predicting the family and then the mechanism within the predicted family reaches 0.106 on exact mechanism against the flat router's 0.116, a paired difference of −0.010 with 95% interval [−0.023, 0.042] and $p = 0.63$: indistinguishable, not better. The inference algorithm of the hierarchical precedent we cite, chained local classifiers with a merged-probability path search (Pan et al., 2023), is lower rather than equal, 0.064 at beam 3

Table 32: The same frozen encoder fed four ways: with and without the <encoder-only> mode-control token its reference implementation prescribes, and pooling over all non-pad positions (what that implementation returns) or over content tokens only. Cause-family accuracy spans 0.038 across the 8 cells. Only a documented invocation is eligible to be the printed floor, a rule fixed before the corrected run; the cell Table 1 prints is marked, and the char-n-gram arm is repeated for reference. `code/baseline_embed.py` writes this table.

Encoder input	over windows	cause	mechanism	printed as the floor
documented	mean	0.253	0.084	**Table 1**, cause
documented	mean ‖ max	0.253	0.096	eligible
content pooling	mean	0.256	0.087	–
content pooling	mean ‖ max	0.253	0.093	–
no mode token	mean	0.253	0.077	–
no mode token	mean ‖ max	0.227	0.090	–
no mode token + content pooling	mean	0.265	0.084	–
no mode token + content pooling	mean ‖ max	0.233	0.096	–
char n-gram TF-IDF, for reference	–	0.238	0.116	**Table 1**, mechanism

Encoder input, hunk ranker	$k = 1$	$k = 3$	printed as the floor
documented	0.198	0.294	eligible
content pooling	0.186	0.294	–
no mode token	0.180	0.299	–
no mode token + content pooling	0.180	0.299	–
largest hunks, heuristic	0.206	0.328	**Table 1**, fix site

(p 0.0102 against a threshold of 0.01022 , which clears its Benjamini–Hochberg threshold at rank 38 of the pooled family, so we report it as a rejection under the whole-paper correction), because the product of a family probability and a within-family probability favours families with fewer children. The 11-way family classifier that §5's closing sentence asks for reaches 0.253 over train and dev and 0.299 on the selection pool where the two models read 0.448 and 0.612 off their own answers, so it clears neither break-even threshold. Two further readings point the same way. Handed the correct family, the trained level-two classifier reaches 0.392, 3.389× the flat router, which is the same shape of result the prompted oracle gives and therefore not an artefact of prompting. And replacing the model's own within-family choice with that trained classifier, while keeping the model as the family router, moves the point estimate down (−0.153 for Sonnet 4.5, $p = 0.032$, which does not clear its threshold of 0.015), so what these data support is that swapping in the trained classifier does not recover the gap, not that it actively hurts. Either way the missing component is the router and not the leaf classifier.

G.6 Injection sweep and instrumentation failures

Measured and replayed points in the injection sweep. Two of the five points on each curve in Figure 1(b) are measurements. At an intermediate injection rate the harness decides per item, from a hash of the item id, whether to hand over the true family; an item that gets it is asked exactly what the family-oracle run asked, and an item that does not is walked exactly as the unaided run walked it, so both requests hit the response cache. All 402 of the 402 interior item-runs return the endpoint's answer, the three interior runs per model report 0 paid calls and 747 cache hits between them, and the curve is therefore two runs reweighted under one mask rather than five measurements. That leaves the crossing with two sources of uncertainty and the earlier version of this paper reported neither. Redrawing the mask with the items held fixed puts break-even at [0.370, 0.582] for Sonnet 4.5 and [0.614, 0.724] for Opus 5 over 10,000 draws; averaging the mask out, which needs only the two endpoints, moves the crossing to 0.460 and 0.666. The other source of uncertainty, resampling items with the mask held fixed, is printed once and printed in Appendix H, from `code/oracle_ci.py`. `code/make_tables.py`, which draws the masks above, bootstraps items too, with a different rule for draws in which the two curves never cross; its interval and the one in Appendix H do not agree

digit for digit, so restating it here would put two intervals on one quantity. Averaging the mask out also reverses which model's shortfall against its own layer-by-layer routing is the larger: 0.236 for Sonnet 4.5 against 0.188 for Opus 5, where the single mask gives 0.167 and 0.205. Redrawing the mask for both models together, Opus 5's shortfall is the larger in 21% of 10,000 draws, 22% on the matched denominators §5 uses, and the difference between the two shortfalls (Opus 5 minus Sonnet 4.5) has median −0.048 over those draws with a 95% spread of [−0.175, 0.059]. Figure 1(b) therefore reports both shortfalls and not their order. Which crossing §5 prints follows from that same mask distribution. The draw the sweep took has 8% of redraws below it for Sonnet 4.5 and 72% for Opus 5, so it is near the bottom of one model's mask distribution and above the middle of the other's, while averaging the mask out lands within 0.003 of the redraw median in all four model-by-axis cells. The crossing of the averaged curve is not the average of the crossings, and 0.003 is how far apart the two are here. So §5 prints the mask-averaged crossing and not the readings in this paragraph. Comparing redraw by redraw, the flat read-off sits below that redraw's own crossing in 57% of draws for Sonnet 4.5 and 98% for Opus 5 on this 67-item routing axis, and in 70% and 50% on the 59-item axis §5 reads, which is the pair that bears on that section: on its own axis the Opus 5 side of the line is a coin flip. The text therefore defends neither crossing, nor the order of the two shortfalls.

The threshold is an expectation over uniform masks, and a deployable router does not draw one. Every crossing above averages over masks drawn the way the harness draws them, uniformly at random from a hash of the item id, and the 10,000 redraws resample inside that same class. A router worth deploying would not draw one: it would hand over the true family on the items it already routes correctly, so its mask is correlated with difficulty. Ordering the injection by the flat read-off router's own per-item family correctness, correct items first, and averaging over the order within each of the two groups, the matched-axis crossing moves to 0.390 [0.339, 0.441] for Sonnet 4.5 and 0.695 [0.644, 0.729] for Opus 5, against 0.469 and 0.664 under uniform masks with redraw bands of [0.386, 0.582] and [0.627, 0.712]. The two models move in opposite directions: the threshold falls for Sonnet 4.5 and rises for Opus 5, so the shift is not a correction anyone can sign in advance. What does not survive the change of mask class is which side of the threshold the flat read-off sits on. Against the selective threshold the read-off clears it by 0.051 for Sonnet 4.5, in 99% of the within-group redraws, and falls short of it by 0.034 for Opus 5, in 17%. Under uniform masks neither model's read-off clears; under this one, 1 of the 2 swept models does. We print the median of the within-group redraws and not the crossing of the group-averaged curve, because the point-estimate rule `code/breakeven_choice.py` enforces, that the averaged curve cross within 0.005 of the redraw median, does not hold in this class: the selective averaged curve crosses at 0.399 and 0.703, up to 0.009 away from the medians above, against 0.003 under uniform masks. The mask class is not a free parameter for the claim §5 rests on. Routing accuracy and exact-mechanism accuracy are both linear in which items are injected, and injecting one item raises either by at most one item's worth, so a greedy order gives the extreme crossings exactly: no injection mask, nested or not, puts the crossing below 0.322 for Sonnet 4.5 or 0.610 for Opus 5, and none puts it above 0.763 and 0.746. That envelope strictly contains the uniform redraw band, so the band we print understates how far the choice of injection mask alone can move this crossing. Walking the graph layer by layer routes 0.237 and 0.475, and that is 0.085 and 0.136 below the lowest crossing any mask can produce, so the shortfall §5 defends holds for every mask and the side the flat read-off falls on holds for none. `code/breakeven_nested.py` reproduces 0.469, 0.664 and both models' routing read-offs from the per-item files before it prints any of the above, and makes no model call.

Detection of an aggregation error. The automatic retire-a-covered-run step originally compared runs by a hash of selected configuration fields. The field list omitted the run tag, the item-set version and the output cap, so three unrelated experiments were treated as reruns of one: a three-vote variant retired the main table, a with-message run retired the with-message ablation, and a 344-item run retired a 67-item one on a different item set. Seventeen of twenty-seven retirements were wrong, and the visible symptom was one field going null in the break-even computation. The replacement rule does not hash a configuration at all: a run is retired only when every configuration field matches and its item set is a strict subset of the replacement's. The aggregation code separately refuses to sum two rows that share an item, which caught two further cases the retirement pass had left.

Four configuration-propagation failures. Each of these produced a cell that looked like a new result and was byte-identical to the baseline. We list them because the pattern is more useful than

any one instance, and because the guard that finally caught them is cheap enough that any harness of this shape should have it.

The first was a group of four graph knobs (label scope, node extras, starting layer and the symptom oracle) added to the argument parser and threaded into the prompt builder for the graph arm only, while the sweep ran them against a code path that had since been refactored. Eight of twelve variants in one overnight sweep were baseline copies. We found it by noticing that four different variants reported the same accuracy to four decimal places.

The second was chain-of-thought and few-shot: the flags reached the prompt builder but the string substitution that inserted them no longer matched, because the surrounding indentation had changed. Same symptom, same detection.

The third was the output cap. The flat arm called the generation helper without passing `max_tokens`, so every flat run used the helper's default regardless of the flag. This one was worse than the others because it did not produce identical cells: it produced a silently capped generation, and a comparison of three cap values that was entirely served from cache.

The fourth was self-consistency voting, and the story we told about it was wrong in two ways that we found only on re-audit. The sampler did receive the vote count: the run made three calls per item. What failed was the scoring: the majority-vote reducer in the search script writes the exact-mechanism and fix-site flags as literal `False` instead of scoring them, because the vote was only ever implemented over cause families. The two zeros the earlier table printed for that row were therefore not measurements. Rebuilt from `runs.jsonl` that variant scores 0.203 on exact mechanism and 0.373 on fix site, and it still loses. The second error was the guard itself: the fingerprint check marks a run invalid when its prompt hash equals the baseline's, which is the right test for a knob that should change the prompt and the wrong one for a knob that changes only how many samples are drawn. Voting three times from the same prompt is supposed to have the baseline's fingerprint. The *invalid* verdict that table carried was a false positive of our own guard.

The guards, in the released harness: the fingerprint stored with a run is the hash of the prompt *actually sent* rather than a recomputation from the arguments; a run whose fingerprint equals the baseline's is marked invalid automatically; the generation configuration actually used is recorded per call and compared against the requested one, with any mismatch raising a warning into the run summary; and every string substitution in the prompt builder asserts that it matched exactly once. The first two would have caught failures one, two and four on the day they were introduced; the third catches failure three, which the first two do not.

Three baseline denominator errors. Our first floors were computed on an earlier 267-item, 40-class version of the corpus and carried forward as constants. Recomputed on the 344 items with the same scorer as the models, the fix-site floor moved from 0.224 to 0.328, which turns Llama-4-Maverick's 0.305 from a clear margin over the old constant into a reading the paired test cannot separate from that floor ($p = 0.38$; of the 64 discordant items 28 go to the model and 36 to the floor, a difference of $[-0.067, 0.023]$). The other 5 models are above the floor and the paired test separates each of them. Figure 1(a) therefore outlines no bar: an outline there marks a cell the paired test puts below its own floor, and on this scoring there is none. Every floor in this paper is now a run in the database, scored by the same code path as every model.

H Statistical detail

Proportions get Wilson score intervals throughout. We do not use normal-approximation intervals: several cells sit near 0.1 with $n < 100$, where the approximation produces intervals that include negative rates.

Two-arm comparisons are paired on items. We report the exact two-sided McNemar test (binomial, not the χ^2 approximation) together with the number of discordant pairs, because with 67 dev items the discordant count is often under 15 and the approximation is not usable there. Where we report a difference of two proportions we also give a paired bootstrap interval over items, 10,000 resamples, resampling items and not answers so that the pairing is preserved. One rule governs every p printed here, in the text, in a table cell and in a caption, and the table generator applies it: every p in this paper is printed to two significant figures, with <0.001 standing for any value below that bound and

>0.99 for any value above it, so a cell carrying more decimal places holds a smaller p and not a more precisely measured one; the ten readings of the Benjamini–Hochberg family's boundary test are the exception, printed to four significant figures because at two significant figures a p and its threshold round to the same string and the margin between them is what that sentence reports.

The layer decomposition is a cross-model average. The decomposition of the graph walk in §5 (the symptom layer right 0.776 of the time, the family right 0.428 of the time given a correct symptom) is computed over 295 graph-arm item-runs, and an item-run is not an item. The pool is the 5 models that ran the graph arm crossed with the graph-arm item set, so every item enters once per model and each rate is an average over models as much as over items. Every paired test in this paper is computed inside one model; these two rates are not, and they should be read with that in view. They describe where the walk fails, not how any one model fails. A Wilson interval on 295 observations would assume them independent, and they are 5 dependent readings of one item set of 59, so we resample items instead: the symptom rate is 0.776 [0.712, 0.841] and the conditional family rate 0.428 [0.329, 0.532]. The design effects are 1.853 and 2.522, so the effective sample sizes are 159 and 91 rather than 295 and its subset, and a Wilson interval on item-runs would have been too narrow by those factors. A per-model split rests on 59 items per model.

Conditioning in the layer-level rates. Reading those two rates as "the walk fails mostly at the second layer" requires care, because the first layer's commitment is irrevocable *by definition of the graph* and not as an empirical tendency. Each of the 32 mechanisms hangs under exactly one family (0 hang under more than one), and each family's symptom parents are exactly its mechanisms' symptom ancestors, so a wrong symptom puts the correct family outside the candidate set on every item, with no exception to find. The count of items where the walk recovered the right family after a wrong symptom is therefore 0, and that recovery is impossible by construction rather than merely unlikely: the zero is not evidence that models fail to recover, and we do not report it as such. The graph is not a tree in the other direction, which is what makes this worth stating: 11 families hang under more than one symptom class, up to 5 of the 6, so one might expect a wrong symptom to leave the right family reachable, and it never does. What the first layer costs is a fan-out: committing to one symptom class narrows 32 mechanisms to between 9 and 30 of them, 28% to 94% of the label set, and the flat arm never makes that commitment. None of this says the annotated symptom is the right one: on 19 of the 295 items with a named symptom (0.064) the gold family is not reachable from that symptom at all (Appendix G.5), so on those items the graph arm is scored against a target no symptom-respecting walk can reach, and the identity above is what guarantees it cannot. Read this paragraph as a statement about the graph's edge set and not as a clean bill of health for the annotation. §5 reports what an arm that skips the layer measures.

We do not correct for multiple comparisons across the 20 variants, and the reason is that no correction is needed to reach the conclusion. Of the 20 variants with a defined paired dev test on cause family, exactly one is significant and it is significantly worse (`S1-startN1`, $p = 0.0063$); among the variants whose point estimate improves at all (one variant), the smallest p is 0.58. No variant was significantly *better* on dev on cause family, with or without correction. On exact mechanism one is: the commit-message arm is 7–0 over the flat call on the 59 items with a leaf gold label ($p = 0.016$), which does not clear its threshold of 0.011 under the whole-paper correction, so we report it and not a rejection. That omission ran in the direction of §4's headline, so we state it here and not in a footnote.

Those 20 dev tests are inside the 186 this paper corrects together, of which 38 survive. An earlier draft left them out: the family held only the 159 p values that appear as named quantities, so 36 tests that appear only as cells of Table 25 were absent from m, and 9 comparisons this paper reports twice, once in the text and once in Table 33, were counted twice. Both errors enlarge every Benjamini–Hochberg threshold, that is, both ran in our favour. This revision removes the aliases that remained, the permutation tests that carried both an older and a current macro name on the same pool and the same value, and moves an alternative statistic of an already-reported test out of the family, so that each test identity occupies one place as the rule above requires. That shrinks m and therefore *loosens* every threshold, the direction that favours us, so we recomputed each retained test against the new cut: no test's verdict changes, and the drop in the surviving count is the removed duplicate itself and not a lost rejection.

The arms of §5 changed the family's composition without changing its size. They removed two paired tests this paper does not report, which the rule above excludes, and added two it does, so m

stays at 186; and because the tests that entered carry smaller p values than the ones that left, and Benjamini–Hochberg steps up, the cutoff moved *out* rather than in. 38 of the 186 survive, and 28 more have $p < 0.05$ uncorrected, so the correction's price over the whole paper is 28 rejections. An earlier version of this paragraph took that price from the earlier family's survivor count instead of from the uncorrected count, and understated it.

Two verdicts have moved with the family. Claude Haiku 4.5 against the strongest zero-call family floor, $p = 0.012$ against a threshold of 0.01, sat inside an earlier and smaller family and does not clear the present one, so the Floors paragraph of §2 reports that comparison as a direction and not as a separation. The hierarchical-inference comparison of Appendix G.5 moved out and then back: p 0.0102 against a threshold of 0.01022 at rank 38, it clears the present threshold. The rule that produced both moves is the same one, and it does not run in one direction only. A test enters the family when the paper prints it, which raises m and shrinks the threshold at every fixed rank; but a printed test with a small p also pushes every larger p down one rank, and Benjamini–Hochberg steps up, so the line can move out. The Haiku comparison lost its rejection inside an earlier family and stays outside this one. The hierarchical comparison came back because two tests entered above it, §4.1's citation-matched fix-site floor test and one of Appendix J's two single-annotator keys, both far enough below it in p to move its rank up faster than the larger m shrank its threshold. We print every test either way and let the line land where it lands; what we do not do is choose which to print after reading its p. The boundary is now the last rejection the whole-paper correction makes: the hierarchical-inference comparison, at rank 38 of the 186 corrected tests, where the exact $p = 0.0102$ sits 1.6×10^{-5} below its threshold of 0.01022, and both sides of it are printed at the finer precision, because the margin is not visible at the two significant figures this paper prints p to. The earlier exchange of §5's arms ran in our favour on both halves and we state it that way rather than as a repair: neither of them reached $p \leq 0.05$ even uncorrected, while the two that entered sit near the cutoff. We do not print the two departing p values here, because printing them would put them back in the family by the rule above and raise m for the sake of a footnote about why they left. What keeps that exchange from being a choice is that the same rule fixes membership before any p is read: a test is in the family if and only if the paper reports it, so the two had to leave once §5 stopped printing them, and keeping them to hold the line down was never available. Pooling costs us one rejection: every test the earlier family rejected is rejected under the pooled one as well, so what the repair moves is the thresholds and not the verdicts, and the comparison an earlier version of this paragraph named as the price of pooling was not one. The test the pooled family now leaves closest to its line is that same Haiku comparison, at rank 39 of the 186, just past the 38 the correction rejects; the per-mechanism SZZ split on the architectural-interaction families is the next one out ($p = 0.014$ against a threshold of 0.011 at rank 40, so we report the direction and not a rejection; Appendix F). The one dev test that does clear its threshold is `S1-startN1` at $p = 0.0063$ against 0.0094, and it is significantly *worse* than the baseline; read instead as the smallest of 19 dev tests, a Bonferroni bound puts it at 0.12, so it is not a stand-alone finding either. We carried one variant forward, supplying the commit message, on its point estimate alone. That is the practice this benchmark argues against, and the choice was deliberate: the follow-up at 344 items tests the practice as well as the variant.

That follow-up gives a split answer, and the split is the finding. On the model the dev estimate came from, the effect reversed. On GPT-5.6 it is positive at $p = 0.026$, nominal and not past correction: it does not survive Benjamini-Hochberg across the 186 corrected tests (rank 50 of 186, threshold 0.013). An effect that were purely a multiplicity artifact would not do the second thing: it would disappear everywhere rather than appear on another vendor's model at five times the sample size. So the correct reading of §4 is neither "20 tests, all negative" nor "one effect found": one input change out of 20 has a real effect on 1 of the 3 models that change was ever run on, and no scaffold effect in this search replicated across models. The search itself ran on Claude Sonnet 4.5 alone, so for most of the 20 "did not replicate" means "was never run on a second model". Two were: the commit message on 3 models (§4), and the plain decision graph on the 5 of 6 ladder models that produced a valid cell (§5).

Sensitivity of the corrected comparison to a second draw. The Sonnet 4.5 flat-versus-graph cause-family row above is one draw of that pair. The other is row 19 of 20 in Table 25, `S01-graph-baseline`: the same 67 dev items, the same two arms and the same metric, with 25 discordant items split 17 to 8 instead of 18 to 3, and $p = 0.11$ instead of $p = 0.0015$. Both draws enter the correction as separate members: the second sits at rank 28 of 186 and clears its threshold of 0.0075, while the first sits at rank 81 and does not clear its own. The runs are E7660 / E1809 and

Table 33: Two-arm comparisons from the main text, paired on items. *A* and *B* are the two arms' accuracies on the items both ran; *only B* counts items B gets right and A does not, *only A* the reverse, and *p* is the exact two-sided McNemar test on those discordant pairs. The graph-versus-flat rows are the ones to read carefully: the flat menu leads on 3 of the 5 models with a valid graph cell and on neither of the 2 weakest, and only Sonnet 4.5 reaches significance on 67 items on its own.

Comparison	metric	*n*	A	B	only B	only A	*p*
flat vs graph (Sonnet 4.5)	family	67	0.448	0.224	3	18	0.0015
flat vs graph (Opus 5)	family	67	0.612	0.478	7	16	0.093
flat vs graph+family oracle (Sonnet 4.5)	mech.	59	0.271	0.542	17	1	<0.001
flat vs graph+family oracle (Opus 5)	mech.	59	0.492	0.763	20	4	0.0015
closed book vs with diff (Sonnet 4.5)	family	344	0.262	0.369	92	55	0.0029
closed book vs with diff (Opus 5)	family	344	0.340	0.584	135	51	<0.001
no message vs with message (Sonnet 4.5)	mech.	311	0.248	0.225	21	28	0.39
no message vs with message (Opus 5)	mech.	311	0.521	0.502	13	19	0.38
no message vs with message (GPT-5.6)	mech.	311	0.334	0.386	31	15	0.026
no message vs with message (GPT-5.6)	family	344	0.453	0.497	29	14	0.032
flat vs graph (GPT-5.6)	family	67	0.463	0.358	9	16	0.23
flat vs graph (Llama-4-Maverick)	family	67	0.313	0.328	6	5	>0.99
flat vs graph (Haiku 4.5)	family	67	0.299	0.299	10	10	>0.99

E9963 / E2853, and the routing rejection of §5 is computed on 2 of the second pair's runs, so it is a second metric on one draw and not an independent replication. A reader who insists the contrast occupy one family slot gets an answer that depends on which draw is dropped, so we give both: dropping the first leaves $m = 185$ with 38 survivors and no other rejections lost, and dropping the second, the draw this rejection rests on, leaves 36 and takes one other rejection with it. We disclose the duplication rather than re-draw, because a third draw would not tell us which of the first two to keep.

Agreement between labelling passes is reported as raw agreement and as Cohen's κ (Landis & Koch, 1977). We report both because κ on a 32-class problem with a skewed marginal is hard to interpret alone, and raw agreement alone flatters a skewed label set.

Uncertainty in the diagnosis–localisation relationship. §4.1 reports a within-model ρ of 0.040 and reads no sign off it. Three measurements say why, and the first is about which cells are in the set at all.

The configuration set. Two decisions fix it, and this ρ is more sensitive to them than to anything statistical. First, a configuration that was run under more than one tag is one point, not two: counting rows gives 30, of which 3 are repeat runs of three configurations. Second, every closed-book cell is out, and that includes the two cells whose view is `closed_book_strict`, which an equality test against `closed_book` lets through. `closed_book_strict` shows no diff, no changed-file list and not even the symptom class, so those two cells cite the fix site on 0 of 86 and 0 of 258 items: zero by construction, which is the whole reason closed-book cells are excluded from this comparison at all. With them in, the within-model ρ is 0.689 and the fix-site axis clears the permutation null below; with them out it is 0.040 and it does not. Every number in this paragraph and in §4.1 excludes them. The distance between the two readings decomposes additively, from 0.015 counting rows with the closed-book cells out to 0.689 counting configurations with them in: 0.024 from one point per configuration, 0.001 from the re-draw at the declared cap, and 0.648 from those two cells.

Both draws. The 22 swept cells were re-run at the 32,000-token cap Setup declares, so 19 configurations of the 27 carry two draws and 8 carry one. Taking the declared-cap draw everywhere gives $\rho = 0.040$ with a bootstrap over configurations of [−0.348, 0.375]; taking the harness-default draw everywhere gives 0.039 [−0.416, 0.426]. The difference is 0.001, and relabelling at random which draw is which inside the configurations that have two does not separate that difference from zero (p >0.99). So the output cap is not what leaves this quantity unresolved; the width of that bootstrap, 0.723 on a coefficient with a range of 2, is. The re-draw does move individual cells: the

mean declared-minus-default shift is −0.003 on cause family and 0.008 on the fix site, with a largest single-cell shift of 0.090.

The permutation null. Permuting item-level outcomes across the (configuration, replicate) slots that scored each item, with the same permutation applied to both axes, the cause-family axis varies more than chance on the declared-cap draw ($p = 0.0043$, which clears its Benjamini–Hochberg threshold of 0.0091) and not on the harness-default draw ($p = 0.064$), so the half of this claim we do assert holds on one draw of the two. The fix-site axis varies more than chance on neither ($p = 0.52$ and 0.30). That is a failure to reject and not an absence: the fix-site axis's between-configuration reliability is zero on this set, so attenuation caps the signal component of ρ at 0.000, and even with both axes' true between-configuration SD set to their 95% upper bounds the cap is 0.399. The rest of ρ here is an offset rather than signal: because the configurations span four different item pools, the ρ that corresponds to no relationship in this design is 0.140 rather than 0, and the observed 0.040 sits below that reference without being separated from it ($P(\rho_{\text{null}} \le \rho_{\text{obs}}) = 0.25$). Nor is the observed value stable: across the 3 slices of this set it runs from −0.248 to 0.040, sign included.

What survives. The asymmetry in effect size: the between-configuration SD is 0.038 for cause family against a 95% upper bound of 0.019 for the fix site, a ratio of 1.995. That ratio is the size of the asymmetry we claim; it is not large, it rests on the draw whose cause-family test clears the correction and not on the other, and it is why §4.1 reports an asymmetry and neither an independence nor a relationship. `code/rho_null.py` produces all of these into `results/rho_null.json`: both draws side by side, the additive decomposition above, the sub-seed of every bootstrap, and a `what_this_does_not_show` field whose first entry is that nothing here shows the fix site is unmoved by prompting.

Intervals on the two oracle multiples and the break-even crossing. The two multiples of §5 are quotients of small counts on 59 items. A paired item bootstrap, 10,000 resamples with one draw of item positions shared by every arm and both models, puts them at 2.00× [1.444, 3.111] and 1.55× [1.212, 2.087]; $P(\text{ratio} < 1.5)$ is 0.04 and 0.38; the first is nominal only and does not clear its Benjamini–Hochberg threshold of 0.017, so after the correction neither multiple can be called clear of 1.5. Their difference is [−0.213, 1.493] and does not clear zero, and the net gain behind them is the same 16 items in both cases. The quantity that is resolved, and the one §5 rests the mechanism on, is the level of exact-mechanism accuracy once the family is given: 0.220 [0.065, 0.373] in Opus 5's favour. This is the one place the paper prints an item interval on the break-even crossing: [0.239, 0.616] and [0.522, 0.836], around the single-mask crossings 0.391 and 0.682 that the sweep's own draw produced. Those are not the thresholds §5 prints, which are the mask-averaged 0.469 and 0.664: neither interval here is centred on the number the main text reads, and the mask uncertainty that separates the two is in Appendix G. The signed margin of the flat read-off router against its own threshold is [−0.119, 0.175] for Sonnet 4.5 and [−0.189, 0.045] for Opus 5. On this same single-mask draw the router §5 defends, the graph's own walk, has a signed margin of −0.167 [−0.403, 0.000] for Sonnet 4.5 and −0.205 [−0.386, −0.026] for Opus 5, and 0 and 0 of the 10,000 resamples put that walk above the routing accuracy this draw's threshold asks for. The mask-averaged shortfall §5 prints is a different number, it has no item interval, and averaging the mask out reverses which model's shortfall is the larger (Appendix G). Sonnet 4.5's upper endpoint here is a clip and not a measurement. The crossing is censored below in 4% of resamples, more than the 2.5% upper tail of a percentile interval, and a censored resample is one in which the graph arm's exact-mechanism accuracy already matches the flat arm's, so the threshold is recorded at the graph's own routing point and the margin there is exactly zero; Opus 5 is censored in 1%, under that tail, which is why its upper endpoint is a negative number. We do not read the resample count as a test and we keep it out of the corrected family: this estimator returns a positive margin only when a resampled injection curve is non-monotone in routing accuracy, so the count is close to a property of the construction. What the interval bounds is the size of the shortfall and not its direction. `code/oracle_ci.py` writes all of these to `results/oracle_ci.json` and refuses to run unless its own point estimates match `results/breakeven.json` to 10^{-9}.

Denominators for the oracle accuracy ratios. The flat arm predicts a cause family too, so the 59 leaf-gold items split by whether it got that family right. Where it did (26 items for Sonnet 4.5, 39 for Opus 5) both arms choose among the same true family's mechanisms and the restricted menu is the only difference: it scores 17 against 16 and 27 against 29, discordant on 3 and 6

items (p >0.99 and 0.69, exact McNemar). Where it did not (33 and 20 items) the flat arm cannot score a mechanism at all, because a wrong family forces a wrong leaf, and the oracle arm scores 15 and 18. The misrouted half therefore supplies 15 and 18 of the 16 net gained items while the correctly-routed half supplies +1 and −2. That decomposition bounds the menu's contribution on the correctly-routed half and cannot bound it on the misrouted half, where the two move together by construction, so the multiple is an upper bound on the family label's value rather than a measurement of it. The family-given column also has a chance floor, which Table 17 now quotes. A true family holds 4.10 mechanisms on average on this pool (2 to 6), and choosing uniformly inside the true family scores 0.294 with no model call. Those two numbers do not divide into each other and the printed floor is the second one: it is the mean of 1/|family| over the 59 items, not the reciprocal of the mean number of leaves, which is smaller because the reciprocal is convex. The floor is `within_family_floor.uniform_expected_accuracy` in `results/oracle_decomp.json`, and it is not 4.10 inverted. Both family-given cells clear that; Sonnet 4.5's flat-menu 0.271 does not, so it is the larger of the two multiples whose denominator sits below the chance level of its own numerator's task. A within-family majority label fitted on train would be a higher floor and we do not compute it. `code/oracle_decomp.py` writes `results/oracle_decomp.json` and refuses to run unless its per-item recount reproduces both arms' `m1_exact` in `runs.jsonl` exactly.

Intervals with matched routing denominators. Those two margins read the routing over all 67 dev items while the threshold that defines them comes from exact-mechanism accuracy over 59, so §5 reads both on the 59 and `code/floor_paired.py` repeats the same paired bootstrap there. The margins become [−0.133, 0.142] and [−0.133, 0.069], both still covering zero; the joint claim that one router clears Sonnet 4.5's threshold and misses Opus 5's falls from 67% of paired resamples to 36%, which is why §5 prints no crossing at all. The graph's own margin barely moves on that axis, to [−0.403, 0.000] and [−0.392, −0.030], and neither axis produces a resample in which the walk clears its own line. Every resample in this paragraph holds the injection mask at the draw the sweep took, and the next figure is conditional on that: the reading that being better at the task raises the threshold faster than it raises this router's accuracy holds in 77% of these resamples and is still not something we report, because averaging that mask out puts the threshold ratio below the router's (Appendix C). §5 therefore reports neither the size nor the direction of that comparison, and the contribution list of §1 says the data do not settle it. The script refuses to print any matched interval unless its own 67-item pass reproduces `results/oracle_ci.json` within Monte Carlo error.

Effective sample size behind the layer decomposition. The symptom and conditional-family rates of §5 are computed over 295 item-runs, which is 59 items read once by each of 5 models. Treating those as independent would be wrong: a cluster bootstrap over items gives a design effect of 1.853 on the symptom rate and 2.522 on the conditional rate, so the effective sample sizes are 159 and 91 rather than 295 and its subset. The pooled gap between the layers survives that correction, 0.348 [0.222, 0.469], but it is not a per-model statement: the symptom rate spans 27.12 points across the 5 models, and the gap's interval excludes zero for 4 of them singly. `code/eff_n.py` writes `results/eff_n.json`.

I Label provenance and the two-pass agreement

The two-pass agreement is measured almost entirely on train. The 252 items carrying both labellings split 238 train to 14 dev, and no test item at all. Table 1 prints that agreement beside accuracies measured on 344 train and dev items, so the two quantities are not on the same pool: one is a train quantity with a dev tail, the other is train and dev in the proportion the splits give. This does not make the agreement wrong, and it is one more reason the paper refuses to read it as headroom (§2): a number measured on the items the taxonomy was induced from cannot bound accuracy on items it was not. The composition is in `probe/ceiling.json` under `on_train_dev_split_mix`, which earlier versions of that file did not record, so the question could not be asked of the artefact.

Every mechanism label in PerfDiag was authored by a model, and models are then scored against those labels. This is the central threat to the benchmark's validity, and the rest of this section gives its size: a two-pass agreement of 0.794 at family level and 0.639 at mechanism level (Table 20), cross-vendor annotator agreement in Table 34, and a human correctness audit of 19 items below.

Provenance of the second labelling pass. The second labelling is the `support` list on each leaf of the taxonomy file: the commits that leaf was induced from. Read backwards it assigns a mechanism to each of those commits, under a prompt whose task was "which commits justify this node" rather than "which node describes this commit". We state that from our own records rather than from a released artifact: unlike the first pass, whose prompt ships verbatim beside its labels, the second pass's prompt was never written to a versioned file, so the sentence you just read is the only description of it and cannot be checked against the bundle. What can be checked is the order: the taxonomy file is timestamped before the first pass ran, and the first pass had no taxonomy to assign from until it existed, so the second pass cannot have seen the first pass's labels. That makes it independent of the labelling *decision* but not of the label *set*, since these are the commits the label set was built to cover, so it bounds how reproducibly a commit can be placed in a fixed taxonomy, not how well the taxonomy was chosen. §2's coverage measurement is the non-circular half of that question and is a separate experiment.

We report the agreement on the scored pool, not on the corpus. Of the 344 scored items, 252 carry both labellings and 92 carry only the card label, so the agreement is measured on 252 items and assumed to hold on the other 92. On those 252 the two passes agree on 0.639 of mechanisms and 0.794 of families (κ = 0.626 and 0.762). Pooling in the corpus commits that are not benchmark items gives 326 pairs and moves the numbers to 0.641 and 0.801, less than a point either way, which is why we print one agreement number in every table rather than two. Neither labelling contradicts itself (0 commits appear under two different mechanisms within one pass), so the disagreement we measure is entirely between the two framings.

A cross-vendor panel. For the 129 items the corpus left unlabelled, 3 models from 2 vendors labelled independently and we kept the majority (two of the three are Anthropic models, so the panel is not 3 independent vendors): 96 of them got one and entered the benchmark, and the other 33 did not. Items where all three disagreed are reported as contested and excluded from the labelled pool rather than resolved by a tiebreak. The panel's agreement with the corpus labels is not read off this pool, which by construction carries no corpus label: it comes from a separate run in which the panel relabelled 51 items that *do* carry a corpus label, agreeing on 0.882 of families.

Table 34: Cross-vendor annotator agreement on the 129 previously unlabelled items, how often each annotator said no label fits, and the ablation that shows the option can fire. Every rate is over the items that annotator answered: Opus 5 returned no tool call on 46 of 129, all of them truncations at the output limit, and those items are excluded here and in the κ rows alike. The bottom block re-asks the same annotators with the menu ablated. The option is live: offering only one distant family drives it to 0.876. It is also a weak coverage claim: deleting the family the annotator itself chose still leaves it a label it accepts on 0.903 of items, so a low rate on the full menu measures the menu's redundancy as much as its coverage.

Annotator pair	n	mechanism	κ	family
Claude Opus 5 vs GPT-5.6	77	0.818	0.805	0.870
Claude Opus 5 vs Claude Sonnet 4.5	76	0.605	0.583	0.724
GPT-5.6 vs Claude Sonnet 4.5	123	0.528	0.503	0.675
"no label fits"	answered	said so	rate	95% CI
GPT-5.6	129	5	0.039	[0.013, 0.088]
Claude Opus 5	83	2	0.024	[0.003, 0.084]
Claude Sonnet 4.5	126	0	0.000	[0.000, 0.029]
menu ablated, same annotators	answered	said so	rate	95% CI
own cause family deleted	72	7	0.097	[0.040, 0.190]
a different family deleted (placebo)	83	0	0.000	[0.000, 0.043]
only one distant family offered	89	78	0.876	[0.790, 0.937]

Sensitivity of the no-label-fits option. A low "no label fits" rate is evidence about coverage only if the option can fire, and Sonnet 4.5's 0.000 over the items it answered is by itself indistinguishable from an option that never fires. We ran the hold-out-one-family control that settles it. For 30

sampled items per annotator we deleted from the menu every mechanism belonging to the family that annotator had itself chosen under the full menu, then re-asked the identical prompt, tool schema and output limit; the full-menu arm replays byte-identical from the cached calls, so the two arms differ only in the menu. None-fits goes from zero on these items, which is how they were sampled, to 0.097. A placebo that instead deletes the non-chosen family with the largest symptom-level overlap leaves it at 0.000 over 83 answered cells: on the 69 items answered under both ablations, 7 move to none-fits when the annotator's own family goes and 0 when a different family does (exact McNemar $p = 0.016$, nominal: its Benjamini–Hochberg threshold is 0.011), so what the placebo shows is that the rise does not reproduce when a comparable but wrong family is deleted. The two menus are not exactly matched in length, and the mismatch runs against us: the placebo menu averages 29.210 leaves against 27.890 when the annotator's own family is deleted, so whatever push a shorter menu gives none-fits acts on the own-family arm, not on the placebo. Offering only the single family with the least symptom-level overlap with the annotator's own, two or three mechanisms in all, drives none-fits to 0.876 over 89 answered cells, and every annotator moves, Sonnet 4.5 included. The option is live.

Four things the control settles against us. With its own family gone an annotator still accepts some other mechanism on 0.903 of items, at self-reported high confidence on 36 of those 65 answers, so a rate near zero on the full menu is consistent with a label set whose mechanisms overlap enough that a substitute is almost always acceptable, and it bounds redundancy as much as coverage. And the corrected rates are wide: the widest 95% Clopper–Pearson interval on the three full-menu rates reaches 0.088, so these data do not exclude a true none-fits rate near one item in eleven. Opus 5 is the weak cell throughout: deleting its own family made it exhaust the output limit without calling the tool on 18 of 30 items, against none of the same items under the full menu, so its arm rests on 12 answers and its zero there carries no weight. And the corrected denominator is not a free repair: the 46 full-menu items Opus 5 never answered are the ones the other two annotators agree on less often. Where they both gave a label they agree on the cause family on 0.595 of those 42 items, against 0.716 of the 81 items Opus 5 did answer, so dropping the unanswered ones measures its rate on the easier part of its own pool. `code/coverage_audit.py` writes `results/coverage_audit.json`.

Sensitivity to mechanism-label granularity. The control above says an annotator accepts some other mechanism on 0.903 of items once its own family is deleted. Every mechanism rate this paper prints is strict single-leaf match against one key, so that 0.903 is an upper bound on how much of the residual mechanism error could be disagreement about which leaf names one mechanism rather than a failed diagnosis, and it is why the span of the mechanism column in Table 1 is not a span of diagnostic skill. Table 35 separates the two on the same 311 items and the same denominator in its five credit columns, with no new model calls, by reusing the cached leaf votes of Appendix J. Crediting any leaf that one of the 5 other annotators also chose on that item raises the six cells to 0.251 to 0.669; counting all 6 votes raises them to 0.338 to 0.675. Table 1's ordering of the six models survives three of the four; under cross-vendor annotators only, Claude Sonnet 5 and GPT-5.6 swap places by 0.3 accuracy points, which is what that column's asymmetry predicts: a model whose own vendor supplied most of the panel has 2 cross-vendor annotators to be credited against, where the other two have 5. What no variant preserves is the size of the span. Best over worst falls from 2.95 under strict match to 2.00 when every annotator's leaf counts, because the weakest model gains most from redundancy, and it rises to 3.69 under the second-choice reading of Appendix C, because the weakest model never fills that field. The symptom column is the same statement at its loosest: 0.962 to 0.994 of answers sit under a symptom the gold leaf also sits under, scored over the 344 items carrying any gold label rather than the 311 carrying a gold leaf, so that layer separates nothing, which is why it is reported as a diagnostic and never as credit (Appendix K).

The top-2 reading against the two-pass agreement. The two-pass agreement of Table 1 is one choice against one choice over 252 items, the second pass never recorded a second choice, so there is no symmetric human top-2, and the two cells as printed do not share a pool. On the 233 items both cover, the second pass agrees with the key on 0.691 against Claude Opus 5's 0.670 under the top-2 reading and 0.558 under strict match. Matched, the second pass stays above the best model under either reading, which reverses the order the unmatched pair of printed cells implies.

Mechanism confusion patterns. Table 37 is the per-leaf confusion summary over the 1,866 model answers: for each of the 32 gold mechanisms, how often the six models named it, and the single

wrong leaf they named most often in its place. That modal wrong answer takes 13% of all answers; on four mechanisms it is more frequent than the gold leaf itself, and one mechanism was never named correctly by any of the six. Table 36 asks the same question of the annotators instead of the scored models. Over the 344 relabelled items, 228 of the 496 leaf pairs were chosen by two annotators on one item at least once, 46%, at a mean rate of 0.050 and at most 0.214. This is the measurement the 94.4% figure of Appendix K is not: that figure counts symptom ancestors in the graph, and the graph relates nearly every pair while the votes confuse fewer than half of them. 192 of the 228 cross a cause-family boundary, so the half-credit family tier does not absorb most of these confusions. On the 162 items all 6 annotators answered, 98 drew at least two distinct leaves, 60%.

Table 35: Redundancy-aware readings of the exact-mechanism column of Table 1, on the same 311 items and the same denominator in the five credit columns, and with no new model calls. *the key*: the printed column, strict single-leaf match. + *2nd choice*: the optional `alternative_node_id` field counts too (Appendix C). + *1 annotator*: the answer counts also when at least one of the 5 other annotators of Appendix J chose that leaf on that item, with the scored model's own vote excluded. + *all 6*: the same with its own vote included, so the gap between these two columns is what self-preference buys under this reading. + *cross-vendor*: only annotators from another vendor count, the strictest of the three and the only one that moves the ordering, because a model whose vendor supplied most of the panel has 2 such annotators against 5 for the other two. *symptom*: `m1_shares_symptom`, whether the answer sits under a symptom the gold leaf also sits under; that flag is defined wherever a gold label exists, so it alone is scored over 344 items and not 311. The last row counts every answer that carried a second choice, over 311 items and 6 models; the per-model counts are 0 for Llama-4-Maverick, 76 for Claude Haiku 4.5, 204 for Claude Sonnet 4.5, 287 for Claude Sonnet 5, 241 for GPT-5.6, 308 for Claude Opus 5.

	Exact mechanism, credited against					
Model	the key	+ 2nd choice	+ 1 annotator	+ all 6	+ cross-vendor	symptom
Llama-4-Maverick	0.177	0.177	0.251	0.338	0.251	0.991
Claude Haiku 4.5	0.215	0.238	0.334	0.370	0.305	0.991
Claude Sonnet 4.5	0.248	0.312	0.373	0.386	0.331	0.985
Claude Sonnet 5	0.363	0.508	0.524	0.524	0.473	0.962
GPT-5.6	0.334	0.498	0.476	0.505	0.476	0.991
Claude Opus 5	0.521	0.653	0.669	0.675	0.630	0.994
best over worst	2.95×	3.69×	2.67×	2.00×	2.51×	–
2nd choice filled	–	1,116	–	–	–	–

Table 37: Per-leaf confusion over the 1,866 answers the six models gave on the scored 311 items, one row per gold mechanism, sorted by how many answers the mechanism carries. *items*: items whose gold label is this mechanism. *exact*: the share of the six models' answers on those items that named it. *Most frequent wrong answer*: the single other leaf named most often on those items, with its share of the same denominator. A row whose rate exceeds its exact column is a mechanism the models name something else more often than they name it.

Gold mechanism	items	exact	Most frequent wrong answer	rate
`gate-reads-wrong-operand`	24	0.181	`unconditional-fallback-pins-slow-path`	0.090
`placeholder-shape-sizes-the-budget`	19	0.325	`redundant-work-deleted-or-gated`	0.114
`redundant-work-deleted-or-gated`	19	0.316	`per-step-device-to-host-readback`	0.114
`device-capability-probe-mispredicated`	17	0.343	`invariant-setup-hoisted-or-memoized`	0.108

continued on the next page

Table 37, continued

Gold mechanism	items	exact	Most frequent wrong answer	rate
per-step-device-to-host-readback	16	0.562	invariant-setup-hoisted-or-memoized	0.115
unconditional-fallback-pins-slow-path	16	0.385	stage-skip-keyed-on-wrong-signal	0.115
invariant-setup-hoisted-or-memoized	15	0.422	redundant-work-deleted-or-gated	0.133
retire-path-release-missing	15	0.267	redundant-work-deleted-or-gated	0.111
default-clobbers-expressed-value	13	0.385	invariant-setup-hoisted-or-memoized	0.077
region-boundary-excludes-executed-work	13	0.244	invariant-setup-hoisted-or-memoized	0.179
captured-artifact-specialized-for-what-the-runtime-never-asks	12	0.250	region-boundary-excludes-executed-work	0.097
consumer-reads-unconnected-config-site	12	0.222	per-element-loop-batched-or-folded	0.111
stage-skip-keyed-on-wrong-signal	11	0.212	per-element-loop-batched-or-folded	0.106
kernel-availability-probe-misresolves	9	0.444	device-capability-probe-mispredicated	0.148
missing-cross-stream-ordering-edge	9	0.296	decision-resolved-once-then-replayed	0.093
operand-form-normalized-before-kernel	9	0.426	kernel-availability-probe-misresolves	0.148
admission-budget-over-wrong-population	7	0.238	per-element-loop-batched-or-folded	0.167
collective-emission-guard-wrong	7	0.214	redundant-work-deleted-or-gated	0.143
cross-rank-rendezvous-condition-divergent	7	0.167	per-step-device-to-host-readback	0.143
allocator-pool-not-reset-at-phase-boundary	6	0.333	invariant-setup-hoisted-or-memoized	0.250
callee-binding-misresolved	6	0.111	invariant-setup-hoisted-or-memoized	0.167
decision-resolved-once-then-replayed	6	0.056	invariant-setup-hoisted-or-memoized	0.278
hook-and-callback-order-vs-state	6	0.222	decision-resolved-once-then-replayed	0.139
launch-geometry-from-wrong-default	6	0.389	unconditional-fallback-pins-slow-path	0.111
per-worker-resource-binding-wrong	6	0.639	redundant-work-deleted-or-gated	0.083
persistent-buffer-binding-or-extent-wrong	6	0.083	per-element-loop-batched-or-folded	0.167
collective-group-membership-wrong	5	0.200	unconditional-fallback-pins-slow-path	0.133

continued on the next page

Table 37, continued

Gold mechanism	items	exact	Most frequent wrong answer	rate
warmup-bound-buffer-reused-against-changed-state	4	0.167	per-element-loop-batched-or-folded	0.125
build-time-arch-specialization-gap	3	0.389	per-element-loop-batched-or-folded	0.167
per-element-loop-batched-or-folded	3	0.611	placeholder-shape-sizes-the-budget	0.167
blocking-handoff-between-concurrent-executors	2	0.917	unconditional-fallback-pins-slow-path	0.083
stream-event-boundary-missing	2	0.000	missing-cross-stream-ordering-edge	0.500

Human audit of label correctness. Two-pass agreement measures reproducibility but cannot detect errors shared by both passes. We assess correctness with a sampling protocol fixed before inspection: equal samples from the two label sources, at least three items from each non-vLLM repository, families sampled in proportion to frequency plus two deliberately rare families, and a recorded seed. For each item, the worksheet shows the inducing commit, the full fix diff and the assigned mechanism with its definition. One author answers whether the mechanism describes what the fix addresses: yes, no or unclear.

Among 19 sampled items in the scored split, the author judges 19 correct, 0 wrong and 0 unclear, spanning 18 mechanisms. The source-specific counts are 15/15 corpus-labelled and 4/4 panel-labelled items.

The audit has five limitations. First, it verifies a visible label rather than eliciting an independent choice among 32 labels; the supplied label can anchor the judgement. Second, the judge is an author rather than an independent expert. Although the author did not write the labels, the study's conclusions depend on accepting them. Third, the sample is small. The 95% lower bound is 0.832, excluding error rates above about 17% but remaining consistent with roughly one error in six.

Fourth, the audit's sensitivity is unmeasured. No worksheet row contains an intentionally incorrect label, and the judge rejects none. We therefore cannot establish whether the procedure would detect a wrong label. By comparison, the link audit uses decoy pairs to estimate false acceptance at 0.008 and tier-A links to estimate sensitivity at 0.967, although both controls bias its correction upwards (Appendix F). The 19/19 result here is an acceptance rate without such calibration. `code/human_check.py` creates no control rows. Testing sensitivity requires a new worksheet with planted sibling-leaf errors and a second judging pass; the existing judgements cannot supply that test. Fifth, the audit does not change the interpretation of two-pass agreement: reproducibility and correctness remain separate quantities.

A score above 0.794 can include agreement with labels disputed by the second pass. It should not be interpreted as independently verified diagnostic accuracy. Table 11 reports corpus and panel labels separately.

Test-label exposure during the audit. 30 items were drawn, not 19. The sampler pooled all 430 items instead of the scored split, so 11 of the drawn items belong to the held-out test split, and their fix diffs and assigned labels were read during the audit (all 11 were judged correct). They are excluded from every number above: no scored quantity in this paper is computed over them. They do enter the macros of the disclosure paragraph below, which counts them. What was seen is label information, not model behaviour: no test-split model output was generated, scored or inspected (the results database refuses to record a test-split run and contains none), so no reported comparison can have been tuned on it. The sampler now draws from the scored split only.

The exposure is concentrated in the test split's less thoroughly audited label source. All 11 are panel-labelled, which is the column whose own bound is the weaker of the two: 14% of the 76 panel-labelled test items have had their key read, against 0 of the 10 corpus-labelled ones. By repository 10 are vLLM and 1 is TransformerEngine, half of that repository's 2-item test cell; the 11 span 9 distinct

Table 36: Vote-derived pairwise confusability of the 32 mechanisms: the 15 pairs two annotators most often chose on the same item, over the 344 items of Appendix J. *both*: items where one annotator chose the left mechanism and another chose the right one. *either*: items where at least one annotator chose at least one of them. *rate*: both over either. *sym.*: whether the pair shares a symptom ancestor in the graph, which 94.4% of all 496 pairs do, so this column is nearly constant by construction and is printed to show that the graph's relation is the weaker statement. 4 of the 228 co-voted pairs share no symptom ancestor at all.

Mechanism	Mechanism chosen instead	both	either	rate	sym.
decision-resolved-once-then-replayed	stage-skip-keyed-on-wrong-signal	18	84	0.214	yes
invariant-setup-hoisted-or-memoized	redundant-work-deleted-or-gated	18	97	0.186	yes
gate-reads-wrong-operand	stage-skip-keyed-on-wrong-signal	13	62	0.210	yes
default-clobbers-expressed-value	gate-reads-wrong-operand	12	61	0.197	yes
decision-resolved-once-then-replayed	unconditional-fallback-pins-slow-path	11	95	0.116	yes
stage-skip-keyed-on-wrong-signal	unconditional-fallback-pins-slow-path	10	70	0.143	yes
decision-resolved-once-then-replayed	region-boundary-excludes-executed-work	10	77	0.130	yes
consumer-reads-unconnected-config-site	default-clobbers-expressed-value	9	47	0.191	yes
decision-resolved-once-then-replayed	device-capability-probe-mispredicated	9	88	0.102	yes
decision-resolved-once-then-replayed	invariant-setup-hoisted-or-memoized	9	114	0.079	yes
callee-binding-misresolved	kernel-availability-probe-misresolves	8	50	0.160	yes
device-capability-probe-mispredicated	kernel-availability-probe-misresolves	8	50	0.160	yes
gate-reads-wrong-operand	unconditional-fallback-pins-slow-path	8	71	0.113	yes
invariant-setup-hoisted-or-memoized	per-step-device-to-host-readback	8	71	0.113	yes
kernel-availability-probe-misresolves	unconditional-fallback-pins-slow-path	7	60	0.117	yes

mechanisms. `code/split_profile.py` writes their item ids to `results/split_profile.json` under `test_split_label_exposure`, and the release carries that file. A later user should report a test number twice, on all 86 items and on the 75 no author has read, and read the gap as the cost of this deviation and not as a property of the models.

The human spot check, split by label source. The pooled bound §2 prints is 19 of 19 correct, Wilson lower bound 0.832. That pooling hides the quantity a reader of the test split most needs. 15 of 15 audited corpus-labelled items were correct, Wilson lower bound 0.796 and Clopper–Pearson 0.782, which excludes a corpus label error rate above 22%. The panel arm is 4 of 4: Wilson lower bound 0.510, exact 0.398, the latter admitting a panel label error rate as high as 60%. §8 prints the Clopper–Pearson bound for both arms, because pairing one method with the other across the comparison widens the printed gap without being conservative on either side: Wilson gives 0.796 against 0.510, Clopper–Pearson 0.782 against 0.398, and the corpus arm's Wilson bound beside the panel arm's exact bound, 0.796 against 0.398, is the widest of the three readings. Four items provide only a weak correctness bound. Because panel labels make up most of the test split, the audit leaves substantial uncertainty about test-label quality. Coverage is 5% of the scored corpus items and 20% of the scored panel ones; the latter looks larger only because the scored panel pool is small.

`code/human_check.py` recomputes every one of these from the 30 hand-entered adjudications, compares them field by field against the derived file and against the 19-item macros, and exits non-zero on any disagreement.

Checks on label circularity. §6 states the problem in one paragraph; here are the numbers behind it. On the 51 dev items of the separate re-labelling run above, the panel agrees with the source corpus on 0.882 of families and 0.824 of mechanisms, which is *higher* than the two-pass agreement printed in every table (0.794 and 0.639), so which pair of passes a reader compares changes the apparent reproducibility by up to 18.50 points, and no one of them is the right one. The two are not measured on the same items either, so that spread is not a difference between two readings of one pool. The panel pool is filtered rather than complete: 33 of 129 candidates were dropped because fewer than two of the three annotators agreed, which removes the hardest items instead of resolving them, so the panel-labelled subset is easier than the population it was drawn from by construction. Table 11 splits every model by label source; the corpus-versus-panel gap there is the size of the effect a reader should discount. That gap is not constant across models, and on the 20 panel items in train and dev its interval is about 0.20, wider than the differences between models, so the per-model panel column must not be compared across models, and §6.1 asks for the split rather than for a comparison of it.

Re-ranking when the answer key is not Claude Opus 5's alone. `code/selfpref.py` re-scores all 6 models on every subset of the scored pool whose label is not Claude Opus 5's word alone, and on one fixed item set with another model's label swapped in as the key. It makes no model calls and scores no test item, and Table 38 prints every subset beside the pooled column. **Three subsets have a key Claude Opus 5 had no hand in, and it is not first on two of them.** On all 20 scored panel items, whose key is a three-annotator majority, it leads at 0.650, above the 0.580 it scores on the 324 items it labelled itself, while beating only 1 of the five others at $p \leq 0.05$. The flip appears on the 7 of those 20 where it returned no vote at all: there it ranks 2 of 6, Claude Haiku 4.5 leading with 0.714 against its 0.571 (exact paired McNemar $p > 0.99$; it ties with 2 others). Those 7 are not one class either: 5 returned no tool call and 2 returned a tool call carrying a null label, and the relabelling records classify the non-vote without recording a finish reason, so we cannot verify item by item that all of them are output-limit truncations.

The 19 hand-checked items are *not* a subset whose key Claude Opus 5 did not write. 15 of them carry its own label and 4 a panel one, the check was not blinded (`protocol.blinded` in the record), and all 30 adjudications (19 of them in the scored split) came back correct with 0 labels replaced. That subset therefore measures accuracy against labels a non-blind author confirmed, most of them Claude Opus 5's own, and there it ranks 3: GPT-5.6 scores 0.737 against its 0.579 ($p = 0.38$). Neither flip is significant and neither subset bounds anything. The Wilson half-widths are 0.20 and 0.30; the 7-item subset is selected by non-votes, so it is not a random sample of the pool; and the hand-checked sample is stratified by repository, with 5 of 19 vLLM items against 212 of 344 pooled. Re-scoring 6 models against a hand-checked label set is a model measurement, not a second human validation: the 30 adjudications are the entire human record and no script can enlarge them. A further subset a reader would ask for, items where Claude Opus 5's own vote lost the panel majority, is *empty*: it voted on 13 of the 20 scored panel items and lost 0 of them, against 1 for GPT-5.6 and 4 for Claude Sonnet 4.5.

Cross-vendor labels without subset selection. Every subset above trades sample size for label independence, and the largest of them is selected by label agreement. One measurement is not: on all 60 re-labelled items, replace the corpus key with GPT-5.6's own label and re-score all 6 models. No item is selected, so the "confirmed items are easier" confound cannot operate here. Claude Opus 5 goes *up*, from 0.617 under its own key to 0.683 on the same items, and stays rank 1; it beats 3 of five at $p \leq 0.05$ (largest $p = 0.14$, and $p = 0.064$ against the model whose key it now is). Swapping in a key written by the other vendor does not lower it, which is the un-confounded form of the question and the one answer here that favours us. Two things stop it settling the matter. GPT-5.6 itself gains 0.050 once its own label is the key, the same circularity running the other way, now measured rather than argued, and 5 of 6 models score higher under the swapped key, so the swap moves the difficulty of the set as well as its author. And on the 11 items where the two keys disagree at family level, Claude Opus 5 ranks 2 at 0.455 behind Claude Sonnet 5 (0.545): the disagreement is where a key's author matters most, and it is also where n is smallest.

Agreement subsets and statistical significance. On the 49 of 60 re-labelled items where GPT-5.6's own vote independently reproduces the corpus family label, Claude Opus 5 is still first at 0.735, ahead of second place by 0.163 against 0.116 pooled, but it beats only 4 of the five others at $p \leq 0.05$ (largest $p = 0.057$), the half-width is 0.12, and all 6 models score higher here than pooled, so this is a subset selected for being easy rather than an unbiased estimate. Requiring both non-Claude Opus 5 annotators leaves 35 items, where it scores 0.800 and beats 3 of five (largest $p = 0.18$). At mechanism level the GPT-5.6-only subset is 44 items and the both-annotator subset 26, and on the latter it beats 2 of five (largest $p = 0.45$): significance goes away as the requirement tightens, which is what a shrinking n does and not a property of the labels. *Cross-vendor* can only mean GPT-5.6, because Claude Sonnet 4.5 shares a vendor with Claude Opus 5. The annotators also saw the *fix* commit, which the evaluated models never see, so a confirming annotation is not the same model solving the item. Self-preference predicts a larger corpus-minus-panel gap for the key's author than for the others; measured, Claude Opus 5's gap is −0.070 and 3 models have a larger one, which is the wrong sign, on 20 panel items, so it is not evidence either. A fourth signature does point the other way and is already printed in Table 20: the agree-minus-disagree gain is largest for Claude Opus 5 (0.242, bootstrap [0.091, 0.389]) and smallest for Claude Sonnet 4.5 (0.038), so the model that wrote the key gains most from its own label being reproducible under a second framing, a reading we had not given that table. The *ordering* of those gains is not itself evidence of authorship: the second largest belongs to Claude Haiku 4.5 (0.178), which ranks 5 pooled, and the gap between the two is 0.063 [−0.106, 0.230].

A summary statistic for annotation sensitivity. On the union of all 80 scored items (23% of the pool) whose label is not Claude Opus 5's word alone, it stays rank 1 and leads Claude Sonnet 5, the pooled runner-up chosen before we looked, by 0.125 [0.025, 0.237] against 0.116 pooled ($p = 0.041$, which does not clear its Benjamini–Hochberg threshold of 0.017). Its lead over the union's own second place, GPT-5.6, is smaller and not significant: 0.113 at $p = 0.078$, and it beats 4 of the five others at $p \leq 0.05$ (largest $p = 0.078$). That does not rule out self-preference. The largest block of the union is selected by label agreement, and label agreement selects easy items; the only items whose key Claude Opus 5 demonstrably did not write number 7, and it is not first on them. Two checks tie these numbers to what the paper already prints: the pooled column of this analysis reproduces the *Cause* column of Table 1 for all 6 models, and the two-pass strata reproduce Table 20 cell for cell; both are assertions in the script, which exits non-zero on any disagreement.

Comparison with human-authored taxonomies. Every handle above stays inside this benchmark: two model passes, a cross-vendor panel, an author audit. One check does not. People categorised performance-bug root causes by hand, in other software, before any of this existed: Jin et al. (2012) name 3 root causes over 109 bugs in server, browser, compiler and database code, and Selakovic & Pradel (2016) name 8 over 98 fixed issues in 16 JavaScript projects. `code/prior_taxonomy_map.py` maps our 32 leaves onto those 11 categories under two readings: strict, where the prior category's published definition names the same defective element, and loose, where it names the same wasted-work pattern whatever part of the system holds the defect. 5 leaves restate a prior category under the strict reading and 18 under the loose one, which contains the strict set; the 14 that match neither carry 132 of the 311 scored items whose gold label reaches a leaf (33 stop at a family). In the other direction 2 of the 11 prior categories have no leaf at all under the loose reading and 6 under the strict one, so the label set is not a superset of what people found either. Both halves count. For 18 leaves a hand-written study of different software reached the same category first, which nothing in our pipeline could have arranged; for the remaining 14 the check is silent, and cannot separate a mechanism specific to AI infrastructure from a label a model invented. Two limits keep it weaker evidence than the two-pass agreement above. Both prior papers predate the labelling model's training data, so a leaf that restates one of their categories may be a memorised restatement and not an independent rediscovery; what the mapping does rule out is a category nobody outside this project has ever needed. And the mapping is a judgement call by one author, with no second annotator and so no agreement number, which is why `results/prior_taxonomy_map.json` records every leaf beside its targets, its one-line reason, and the prior definitions copied from Table 2 of Jin et al. (2012) and Section 3 of Selakovic & Pradel (2016), with the URL and SHA-256 of both sources: a reader who disagrees can re-decide a row and recount.

Part of the first gap, the leaves with no prior analogue, is a population difference. The prior corpora are server, browser, compiler and database code (Jin et al., 2012) and client-side and server-side

Table 38: Every subset of the scored pool whose answer key is not Claude Opus 5's word alone, re-scored for all 6 models at cause-family level, beside the pooled row, the two Claude Opus 5-key rows kept for comparison (the 324 items it labelled itself, and the re-labelled pool under its corpus key), and the two rows that swap GPT-5.6's own label in as the key on that same pool. Columns: Llama is Llama-4-Maverick, Haiku is Claude Haiku 4.5, S 4.5 and S 5 are Claude Sonnet 4.5 and Claude Sonnet 5, GPT is GPT-5.6 and Opus is Claude Opus 5; *rank* is Claude Opus 5's competition rank in the row (one plus the number of models strictly above it, so ties are reported as ties); <.05 is how many of the five other models it beats at $p \leq 0.05$ under an exact paired McNemar test; $p_{\max}$ is the largest of those five p values. Indented rows are subsets of the row above. No row is a random sample of the pool and no row bounds anything: the two smallest are selected by non-votes and by key disagreement, the agreement rows are selected for items an annotator confirmed, and every row with n below 49 has a Wilson half-width above 0.12. `code/selfpref.py` writes `results/selfpref.json` with the item ids of every row.

Subset	n	Llama	Haiku	S 4.5	S 5	GPT	Opus	rank	<.05	$p_{\max}$
all scored items	344	0.317	0.337	0.369	0.468	0.453	0.584	1	5	0.00
key by Opus alone	324	0.321	0.321	0.358	0.466	0.451	0.580	1	5	0.00
key = panel majority	20	0.250	0.600	0.550	0.500	0.500	0.650	1	1	1.00
Opus did not vote	7	0.143	0.714	0.571	0.571	0.429	0.571	2	0	1.00
human-checked (non-blind)	19	0.421	0.368	0.474	0.632	0.737	0.579	3	0	1.00
key confirmed by GPT	49	0.429	0.388	0.510	0.571	0.571	0.735	1	4	0.06
confirmed by both non-authors	35	0.486	0.400	0.571	0.657	0.600	0.800	1	3	0.18
relabel pool, corpus key	60	0.350	0.317	0.467	0.483	0.483	0.617	1	3	0.08
relabel pool, GPT's own key	60	0.400	0.383	0.450	0.567	0.533	0.683	1	3	0.14
the two keys disagree	11	0.273	0.364	0.182	0.545	0.364	0.455	2	0	1.00
union: key not Opus's alone	80	0.400	0.438	0.512	0.575	0.588	0.700	1	4	0.08

JavaScript (Selakovic & Pradel, 2016), so a collective or capture-region leaf of ours has nothing to match, and `results/prior_taxonomy_map.json` records that argument under `limits`. It does not carry over to the other direction: the 6 prior categories with no leaf are general API-usage and redundant-work patterns, so their absence is a gap in our label set and not a difference in corpora. The same file records why Jin et al. (2012)'s three non-root-cause axes stay out of the mapping: pooling them would absorb leaves through their API and parameter rows and raise the match count without a root-cause claim behind it.

Both gaps are named row by row rather than counted. Table 39 lists all 14 leaves with no prior analogue under either reading, each with the item count it carries and the one-line reason already published in `results/prior_taxonomy_map.json`, and Table 40 lists the 6 prior categories with no leaf of ours together with their published glosses and which of them the loose reading still leaves empty. A reviewer deciding whether a leaf is specific to AI infrastructure or is a label a model invented has to read those rows, and until now this paper named none of them.

Table 39: The 14 mechanisms with no analogue in the 11 hand-written root-cause categories, under either reading, named row by row with the reason recorded in `results/prior_taxonomy_map.json` at `per_leaf[].why`. They carry 132 of the 311 scored items whose gold label reaches a leaf, so this is not a tail. *Items* is train and dev only; the 86 test items are not counted. Every row is one author's judgement with no second annotator, and a reader who disagrees can re-decide a row from that file and recount.

Mechanism	Items	Why no prior category matches
`placeholder-shape-sizes-the-budget`	19	A one-shot sizing expression is evaluated from a placeholder extent or the wrong memory baseline, so the buffer is sized for work that does not exist. No prior category concerns sizing expressions.

Table 39, continued

Mechanism	Items	Why no prior category matches
device-capability-probe-mispredicated	17	A run-time capability boolean is read, parsed, scaled or compared wrongly and the wrong branch is taken for the whole run. Jin et al. place misunderstanding of platform and API behaviour on their 'how bugs are introduced' axis, not among root causes.
retire-path-release-missing	15	The one code path that retires a unit of work returns without dropping a reference, so everything reachable through it stays resident. Neither prior set has a category for a missing release.
default-clobbers-expressed-value	13	A default or re-derivation on the same field executes after the caller's value and supersedes it. The resolution order of a configuration field appears in neither prior set.
region-boundary-excludes-executed-work	13	Executed work sits outside the compiled or captured region, and the diff changes which code the compiler ever sees. Compiled-region coverage has no analogue in either prior set.
consumer-reads-unconnected-config-site	12	The field the machine reads is not the one the caller's value was written to. Configuration plumbing appears in neither prior category set.
admission-budget-over-wrong-population	7	A running count that a scheduler or allocator consults is accumulated over the wrong set of items. No prior category concerns accounting quantities; the nearest prior slot is the residual 'Others' row of Jin et al.'s Table 2.
allocator-pool-not-reset-at-phase-boundary	6	Storage stays checked out of the caching allocator past the phase that was supposed to bound it. Neither prior set has a category for memory held too long: both study wall-clock cost in single-process programs.
decision-resolved-once-then-replayed	6	A dispatch or control-flow decision is resolved once and replayed against changed state. This is the inverse of 'Repeated execution of the same operations', whose fix is to memoize: here memoizing is the defect, so the prior category does not contain it.
launch-geometry-from-wrong-default	6	A scalar launch-configuration value is resolved from a wrong default, threshold or tuned table. Jin et al. have 'Change A Parameter' as a fix strategy on a different axis; no prior root-cause category covers it.
per-worker-resource-binding-wrong	6	The scope at which a worker derives the physical resource it claims is wrong, so co-located processes collide. No prior analogue: both prior populations are single-process or single-machine.
collective-group-membership-wrong	5	The transfer reaches the wrong set of ranks. Its result is used, by the wrong participants, so 'Skippable Function' would misdescribe it, and communicator scope appears in neither prior set.
warmup-bound-buffer-reused-against-changed-state	4	A buffer bound once in warm-up, profiling or capture is reused under an extent, layout or liveness that no longer matches. Like the replayed-decision leaf this inverts the prior 'repeated operations' category: binding once and reusing is the prior fix and the defect here.
build-time-arch-specialization-gap	3	The defective value is frozen when the binary or extension is produced. No prior category is about build-time specialization; Jin et al.'s 'API Issues' row would take it if the introduction axis were pooled with the root-cause axis, which we do not do.

Table 40: The other direction: the 6 of 11 hand-written root-cause categories with no leaf of ours under the strict reading, 2 under the loose one, with each category's published gloss. The label set is therefore not a superset of what people found. The population argument that covers the other direction does not cover this one: these are general API-usage and redundant-work patterns, so their absence is a gap in our label set rather than a difference in corpora.

Prior category	Published gloss	Loose
Uncoordinated Functions (`jin.uncoordinated`)	function calls take a detour to generate results	filled
Inefficient API usage (`sp.api-usage`)	the code does not use the API in the most efficient way to achieve its goal	filled
Generic API is inefficient (`sp.generic-api`)	an existing API provides a more generic, and therefore less efficient, functionality than required by the program	filled
Inefficient reimplementation (`sp.reimplementation`)	the program implements some functionality that is already implemented in a more efficient way, e.g., as part of the built-in API	still empty
Repeated checks of the same condition (`sp.repeated-checks`)	the program repeatedly checks the same condition, even though some of the checks could be avoided	still empty
A computation can be simplified or avoided in special cases (`sp.special-case`)	code performs a computation that is unnecessarily complex in some special case	filled

J Independent relabelling of every scored item, and re-scoring against a key the scored model did not help write

Appendix I bounds label reproducibility with a second pass written by the same model that wrote the key. This section replaces that with independent annotators. All 6 evaluated models relabelled all 344 scored items: each saw the inducing commit diff, the fix commit diff and both commit messages, chose one of the 32 leaf labels from the full menu, and never saw the existing label. Every call was already in the response cache, so this section costs no new model calls and reads no test item. **Every number here is model-annotated.** The only human record in this paper is the 30-adjudication audit of Appendix I, of which the 19 that fall in the scored split are the ones that give the 0.832 lower bound; nothing in this section is evidence of the same kind, and nothing here may be pooled with it or printed beside it as though it were.

Information available to relabelling annotators. The annotators see the fix commit. The models in Table 1 see only the inducing diff and no commit message. So an annotator agreeing with the key is not the same event as a model solving that item, and we never combine the two sets of numbers. The measurement also cannot separate self-preference from annotator strength: the order of the 6 annotators by agreement with the key is the same order as the benchmark ranking, Spearman $\rho = 1.000$. A stronger annotator should reproduce a correct label more often whoever wrote it, so agreement alone cannot size the confound. The re-scoring below can, because it changes the key and not the items.

Agreement with the existing key, per annotator. Table 41 gives the rates. Family agreement is over the items that annotator answered; mechanism agreement drops the 33 items whose key stops at a family node, so its denominator is at most 311. The denominators are not comparable across rows: answered rates run from 0.54 to one, and the two annotators that answered least are the two with the highest agreement. Claude Opus 5 answered 253 of 344 and reproduces its own key on 0.917 of families (95% Wilson $[0.876, 0.945]$) and 0.891 of mechanisms; if every item it did not answer is counted as a disagreement the family figure falls to 0.674. Split by provenance, it reproduces the key on 0.912 of the corpus-labelled items it answered and 1.000 of the 13 panel-labelled ones, and the

panel arm is not a clean control either: those keys are themselves a majority vote of three annotators, two of which are evaluated models.

The self-preference contrast. We wrote this criterion down and committed it before reading any of these labels: self-preference is evidenced if Claude Opus 5 agrees with its own key significantly more often than every other annotator, paired on the items both answered. It does not. Against the strongest independent annotator, Claude Sonnet 5, the gap is 0.013 over 157 paired corpus-labelled items (exact McNemar $p = 0.62$), and against the strongest annotator from a different vendor, GPT-5.6, it is 0.029 ($p = 0.092$). Only 3 of the other annotators are beaten at $p \leq 0.05$, and they are the weakest models in the pool: the largest gap, 0.277, is against Llama-4-Maverick. By the wording fixed in advance, this is the branch in which self-preference is bounded rather than shown.

Assessing two-pass agreement. Appendix I prints 0.794 and 0.639, which are one annotator's self-consistency across two framings. On the 230 items where both readings exist, that self-consistency is 0.791 at family level while the majority of the annotators other than Claude Opus 5 agrees with the same key on 0.761 (exact McNemar $p = 0.46$). At mechanism level, on 210 items, it is 0.686 against 0.695 ($p = 0.90$). The two readings are not distinguishable, so replacing the self-consistency number with one from independent annotators does not lower it. Over all items where it is defined, the majority of the other annotators agrees with the key on 0.775 of families (315 items) and 0.714 of mechanisms (283 items); using all 6 votes gives 0.812 over 319 items.

Unmeasured identifiability from the inducing diff. The annotators read more than the fix commit: `code/relabel.py` puts the message of both commits in the same prompt, and a scored run gets neither the fix commit nor either message. So 0.775 of families and 0.714 of mechanisms are not an upper bound on what a model could reach from the inducing diff, and they are optimistic even as annotator levels: they condition on a strict majority existing, which at family level holds on 315 of 344 items, and items with no majority are harder, which is what the re-scoring below reports for its own dropped set when 5 of the models score above their Table 1 value on what remains. What no measurement in this paper reports is identifiability, whether the inducing diff alone determines the label at all. That needs annotators from outside this project working from the prompt the scored models get. Until it exists, the distance between these rates and Table 1 is an information gap and not headroom, which is why §8 says "hard" is about the models.

The strictly cross-vendor reading. Only Llama-4-Maverick and GPT-5.6 come from vendors other than the key author's. Their rates are in Table 41. Where the two agree with each other, on 195 of the 341 items both answered, that consensus matches the key on 0.892 of families. That figure is not an unbiased estimate: it conditions on the two annotators agreeing, and items on which independent annotators agree are easier by construction. It says where cross-vendor consensus lands, not how accurate the key is.

Re-scoring against a key the scored model did not help write. This is the measurement that answers the overlap objection without selecting items. For each evaluated model we build a cause-family key from the votes of the other annotators only, so no model is ever scored against a key it helped write, and we re-score the same predictions Table 1 is built from. The unit of independence here is the model and not the vendor: four of the six from one vendor, so for Claude Opus 5 the key is the unique majority of the five other annotators, three of which share its vendor, and two votes are enough to raise a label, so a same-vendor pair can carry a key on its own. The paragraph below replaces the voting key with a single cross-vendor annotator's own labels to remove that. The key exists on 275 of 344 items; the 69 dropped items are those where some key has no majority, and they are harder, which is why 5 models score above their Table 1 value here. So the readable quantities are the ranking and the within-item difference, not the level. Claude Opus 5 ranks 1 under this key at 0.644, against 0.625 under the printed key on the same items, a change of 0.018. The whole order is the Table 1 order. The paired tests use a key that excludes both models being compared, so both are scored against the same key: Claude Opus 5 is above all 5 of the others at $p \leq 0.05$, the largest of those p-values being 0.0037 against Claude Sonnet 5 at 0.553. Self-preference predicts that the key's author loses the most when the key is replaced; the model that gains least is Claude Sonnet 4.5. Voting on mechanisms instead of families and mapping up gives 0.668 for Claude Opus 5 over 235

items, still rank 1, but the order is not identical: Claude Haiku 4.5 and Llama-4-Maverick change places. We print both vote levels.

Selection bias from missing labels. Of 259 annotator-item cells with no usable label, 257 have a recorded stop reason of output-limit truncation at the 1,200-token cap this run requested; the remainder returned a tool call with a malformed argument. A missing label is therefore mostly a harness limit and not an annotator declining. It is also not missing at random. Claude Opus 5 is missing on 91 items, and on those items a mean of 1.879 of the 6 evaluated models solve the cause family, against 2.763 on the items it did answer (rank sum p <0.001). Its missing items also have longer inducing diffs (p = 0.017, nominal: its Benjamini–Hochberg threshold is 0.012), while label provenance is unrelated (p = 0.43, exact Fisher). Claude Sonnet 5, missing on 158 items, shows the same difficulty pattern (p <0.001). Both annotators therefore answered a systematically easier subset, so every agreement rate computed on answered items only, including 0.917 and 0.909, overstates what it would be on all 344 items. The re-scoring paragraph is not biased in the same way: it scores model predictions over a fixed item set, and its paired tests use keys that exclude both compared models.

Items with no majority. 25 of the 344 items have no

A key from a single cross-vendor annotator. The key above is a vote, so the same-vendor majority it can contain is a residual worry. Replacing the vote with one cross-vendor annotator's own labels removes it, at the cost of that annotator's noise going in unaveraged. Against GPT-5.6's labels alone, on 343 of 344 items with 1 dropped for want of a label, Claude Opus 5 ranks 1 at 0.609, it is above 5 of the other five at $p \leq 0.05$ with the largest of those p values p <0.001 against a threshold of 0.0051 against Claude Sonnet 5, and the order is the Table 1 order exactly (Claude Opus 5, Claude Sonnet 5, GPT-5.6, Claude Sonnet 4.5, Claude Haiku 4.5, Llama-4-Maverick). Against Llama-4-Maverick's labels alone, on 342 items, Claude Opus 5 still ranks 1 at 0.439, but the rest of the order moves Claude Haiku 4.5, Claude Sonnet 4.5, GPT-5.6, Llama-4-Maverick (Claude Opus 5, Claude Sonnet 5, Claude Haiku 4.5, Llama-4-Maverick, GPT-5.6, Claude Sonnet 4.5) and the lead over Claude Sonnet 5 is not separated, p = 0.06 against a threshold of 0.019. So first place survives a key written by a single annotator from another vendor and the ranking below it does not. Two readings cut against us. In each key the annotator that wrote it scores against its own labels, which is circular by construction and is reported as such: GPT-5.6 reaches 0.496 at rank 3 under its own key, 0.041 above its value under the released key on the same items, and Llama-4-Maverick reaches 0.336, 0.020 above. And under both keys the model that gains most from the switch is the key's own author and the key's own author, which is what self-preference predicts. All labels were cached, so this arm cost no model calls, and `code/indep_relabel.py` writes it under `solo_cross_vendor_key`. We do not report an unconditional two-annotator majority, because a strict majority of two exists exactly when the two agree, which is the 195-item consensus subset above and is selected by agreement.

unique family label with at least two votes. They keep their per-annotator rows, since having no majority is a property of the panel and not of one annotator, and we drop them whole from any quantity whose key is a majority vote and print the dropped count beside it. We do not break ties: a tie-break by annotator order would make the label depend on the order we happened to list the models in.

Table 41: Independent relabelling of the 344 scored items, and re-scoring against a key the scored model did not vote on. **All annotators are models**; no cell here is a human judgement, and none of it may be read beside the human bound 0.832 of Appendix I. Left block, each model as an annotator against the existing key: *n* is the items it answered, family is over that *n*, and mechanism drops the 33 items whose key stops at a family node. The denominators differ by row and are not comparable. Right block, cause family for the same predictions as Table 1 on the 275 items where every key is defined: printed is the released key, indep. is the majority of the annotators other than the model being scored, and rank is under indep. The last three rows carry their own denominators: the majority of the other annotators against the key over 315 items (mechanism column 283), and the two-pass agreement of Appendix I against the independent reading on the same 230 items (mechanism column 210). Generated by `code/indep_relabel.py`.

	As annotator, vs the key			As evaluated model, cause family			
Model	*n*	family	mech.	printed	indep.	Δ	rank
Llama-4-Maverick	342	0.564	0.395	0.338	0.353	+0.015	6
Claude Haiku 4.5	344	0.640	0.514	0.378	0.385	+0.007	5
Claude Sonnet 4.5	337	0.700	0.583	0.411	0.411	+0.000	4
Claude Sonnet 5	186	0.909	0.855	0.520	0.553	+0.033	2
GPT-5.6	343	0.784	0.755	0.491	0.509	+0.018	3
Claude Opus 5	253	0.917	0.891	0.625	0.644	+0.018	1
majority of the other 5	315	0.775	0.714	–	–	–	–
two-pass agreement	230	0.791	0.686	–	–	–	–
same items, independent	230	0.761	0.695	–	–	–	–

K Prompts and qualitative examples

The flat prompt, verbatim. The system message states the task, that the repository is an AI-infrastructure project, that the commit shown introduced a performance regression, and that the answer must be one label from the list plus a list of (file, line, symbol) sites plus one discriminating prediction. The user message contains the repository name, the author date, the changed-file list and the unified diff, followed by the 32 labels each with a one-sentence definition. No example answer is given in the default configuration.

```
SYSTEM:
You are a performance engineer doing root-cause analysis
on a commit in a large GPU training/inference framework.
You reason about *why* a change costs time or memory,
not about code style. You never claim to have run
anything: no profiler, no benchmark, no test output is
available to you. Your evidence is the code as written.

TASK (diagnostic framing, used for M1/M2/M3):
The commit below was later found to have introduced a
performance regression: after it landed, the system got
measurably worse on at least one of throughput, latency,
memory footprint, GPU utilisation, liveness (a hang), or
compile/startup time. The regression was fixed by a
later commit that you cannot see.

Your job: name the mechanism by which THIS commit causes
that regression, point at the exact code that has to
change to fix it, and state a pair of predictions that
would tell a colleague whether you are right.

TASK (neutral framing: decoys and the FPR control):
The commit below is from the history of a large GPU
```

```
training/inference framework. It may or may not have
introduced a performance regression -- a measurable
worsening of throughput, latency, memory footprint, GPU
utilisation, liveness (a hang), or compile/startup time
that a later commit had to undo. Most commits do not.

Decide first whether this commit introduces such a
regression. If it does not, set 'introduces_regression'
to false and stop -- do not invent a mechanism. If it
does, name the mechanism, point at the code that must
change, and give the two predictions.
```

The two framings differ in exactly one thing: whether the model is told a regression is present. This matters more than it looks. Every recall number in the main text is measured under the diagnostic framing, and every false-positive number under the neutral framing, and the real items were *re-run* under the neutral framing so that the two sides of the false-positive comparison share a framing. Without that re-run the recall and the false-positive rate are measured under different instructions and cannot be put in the same sentence. Table 21 is that re-run, one row per model with the output cap matched across models.

Answer normalisation. Models frequently return a label followed by its definition ("`N2.gate-reads-wrong-operand: a guard evaluates ...`"). Strict string matching recorded most answers as unparseable and cut flat-arm family accuracy by more than two thirds, a scoring artifact rather than a result. The scorer now matches the longest label prefix and strips trailing prose. Any benchmark of this shape needs this; we mention it because our first numbers were wrong in this direction and the error was large.

Credit tiers. Mechanism credit has exactly three levels: exact label 1.0, correct family with wrong mechanism 0.5, otherwise 0. An earlier version awarded partial credit for sharing a symptom-layer ancestor. That tier was free: of the 496 pairs of the 32 mechanisms, 94.4% share at least one symptom ancestor, while the annotators' votes ever confuse only 228 of those pairs (Table 36). Symptom-layer agreement is reported as a separate diagnostic column in Table 35, where it runs from 0.962 to 0.994, and does not enter credit. The paper reports these levels as two binary rates, the exact-mechanism rate and the cause-family rate, and never as the graded score: an exact label always sits in the correct family, so the mean credit is the midpoint of those two rates and carries nothing they do not. The graded value is recorded per item as `m1_credit`, and no printed number averages it. Cite the exact-mechanism rate for a mechanism claim, with the cause-family rate beside it.

A representative failure. On a Megatron-LM commit that moved a device-capability probe inside the per-microbatch loop, five of six models named a memory-allocation mechanism and cited the allocator file. The fix commit hoists the probe. The models identified that the commit added per-iteration work, so the symptom layer was right, and then chose the wrong family, which is the irrecoverable error of §5 appearing in the flat arm as well.

L Measurements for the seven reporting requirements

§6.1 states the seven requirements without their evidence. Each one below carries the measurement it rests on and the number that measurement returned. **Report cause family and exact mechanism separately, and the fix site at line level with its floor**: holding the model at Claude Sonnet 4.5 and varying only the prompt over 27 configurations, cause-family accuracy moves with a between-configuration SD of 0.038 while the fix site moves by at most a 95% upper bound of 0.019, a bound and not a measured absence (§4.1), and the fix-site column sits near a heuristic that reads no code, 0.328 for the heuristic against 0.305 for the weakest of the six, and that comparison has to be made at a matched citation budget: the heuristic cites three sites where the weakest row cites 1.40, and held to one citation the heuristic scores 0.206, which that row clears (p <0.001 against a threshold of 0.0038, 51 items to 17 of 68 discordant). The line criterion is the site metric we designate primary, on the ground that it is the only one of the 4 with a zero-call floor measured beside it; a report quoting one of the other 3 has to say which, because they do not order the models the same way (Table 29,

Appendix G.4). **Report the answer shape**: the fix-site metric is defined over a cited site list, and an answer that parses but serialises that list as a string scores zero on all three fix-site metrics without being unparseable. That is 24 of Table 1's answers, all from Claude Sonnet 5 and Claude Haiku 4.5 (16 and 8), so 2 of the six rows are not on the footing the rest are, and 13 of them cannot be recovered by any parse (Appendix C). The same field carries the citation budget requirement one turns on: the per-answer mean runs 1.40 to 2.91, a factor of 2.08. **Report the label-source split, and the repository mix beside it**: the gap between the corpus-labelled and panel-labelled items is not constant across models (Table 11), and the label sources do not carry the same evidence, the human correctness bound being 0.782 on the 15 corpus-labelled items audited against 0.398 on the 4 panel-labelled ones, both Clopper–Pearson and both the numbers §8 prints (Wilson gives 0.796 and 0.510; §2 prints the panel arm's exact bound only). The repository mix is the second axis and it moves with the first: 62% of the 344 items scored here are vLLM against 87% of the test split (Table 9; Table 12 splits the same runs by repository instead), and the panel column is where the annotator overlap of §2 would show up, so pooling hides both. **State the output cap and both truncation rates**: a binding cap silently converts a score into a lower bound and the parse gate does not always catch it, at 4,096 tokens Opus 5 truncated 16 of 32 calls and still parsed at 0.919. The second rate is on the input: 24% of the 344 scored items were answered from an inducing diff clipped at 900 lines (82 items, 108 of the 430 released), and per-model fix site on them runs 0.146 to 0.296 against 0.351 to 0.659 on the rest, a gap that mixes the harness with item difficulty and that we print no test for (Table 2). **Report tokens with accuracy**: the cost-quality Pareto front here is 4 deep (Figure 6), so "best" alone is not a ranking. **Report the framing**: every number here except the decoy rates is measured with the model told a regression exists, and under neutral framing recall minus the clean-decoy false-positive rate is 0.281 to 0.447 over the 3 models tested (0.281 [0.159, 0.385] for Sonnet 4.5, 0.301 [0.185, 0.415] for GPT-5.6 and 0.447 [0.321, 0.553] for Opus 5, Newcombe 95%: the three intervals overlap but each clears zero, and each is a lower bound, the decoy rates being upper bounds), so a number measured under one framing says nothing about the other (§6). **Report accuracy by self-reported confidence**: the `confidence` field every answer carries separates correct from incorrect answers for 3 of the 6 models and not for the other 3, so answering only where Claude Opus 5 reports high confidence gives 0.789 on 22% of the queue against 0.584 on all of it, while the same restriction buys nothing on the models whose field is saturated or left empty (Table 7).

M Extended related work

The main text places PerfDiag against ten prior works in one paragraph. This appendix discusses all of them, the closest one in a paragraph of its own, then adds four further groupings, some of whose works the main text cites only in passing. We check every statement below about a cited work against that work's published abstract, cached by `code/fetch_relwork.py`; where the abstract does not settle a question we say so rather than fill it in.

Comparison with EffiHolmes. EffiHolmes (Yang et al., 2026) shares the construction this paper uses: it scores its output against a place a human changed rather than against a runtime measurement. ISO-Bench (Nangia et al., 2026) also scores against human patches, but only as one of two metrics and with the bottleneck handed to the agent, so EffiHolmes is the closer of the two. It localises the fix for a reported time inefficiency at repository scale on RepoEffi-Bench, 140 issues collected from popular Python repositories, and reports Acc@3 over files and Acc@5 over functions. Three differences decide what is left over. Its evidence is a differential profile: it runs the repository under a default and a scaled workload and compares the two, which is the step our construction rules out. We execute nothing, for the reason §1 gives: the workloads that expose regressions in our four repositories need multi-GPU hardware and a harness those repositories do not ship. Its input already names the inefficient function, which the issue report supplies, so its question starts after someone has tied the symptom to a code region, while our input is the diff of the commit that introduced the regression and nothing else. And it announces no mechanism taxonomy: the answer is a location, not a name for what went wrong. The consequence for us is the unfavourable one. Our M2 measures the same thing EffiHolmes already measures, so we do not claim the fix site as a new scored output, and we quote it only with the 0.328 zero-call floor beside it. A file-granularity reading is the one axis the two constructions share, and it is the one to compare on: EffiHolmes reports file-level Acc@3 and function-level Acc@5, and our file-level rate runs 0.672 to 0.875 over 344 items (Table 29). The levels are still not comparable and we do not compare them. Its published abstract gives only

the improvement over its baselines and not the absolute Acc@3, our items are commits rather than issue reports, and its input already names the inefficient function while ours does not, so the shared axis fixes the granularity of the question and nothing else. What is left that EffiHolmes does not carry is M1, the mechanism label, and it arrives with a two-pass agreement, a zero-call floor and a label-source split attached.

Relation to fault localisation. M2 is a fault-localisation measurement and we should say whose conventions it borrows. Spectrum-based localisation ranks program elements by a suspiciousness coefficient computed from passing and failing test coverage and reports how often the faulty element falls in the top k (Jones & Harrold, 2005; Abreu et al., 2006), a line surveyed in Wong et al. (2016); the k that matters is small, because developers stop inspecting early and a technique that is right only deep in its ranking does not help them (Parnin & Orso, 2011; Kochhar et al., 2016). Two things that setting has we do not: a failing test to compute coverage from, whose absence §1 builds the task on, and a single faulty element, where a performance fix can touch several files. The recent execution-free work scores a set of candidate locations at file and function granularity instead (Xia et al., 2024; Chen et al., 2025b; Yang et al., 2026). M2 sits between the two conventions and is closer to the second. It is not a ranked list scored at several k: the answer is an unranked set of at most 3 sites, credit is a hit anywhere in the set, and the effective k is whatever the model chose to cite, 1.26 to 2.91 files per answer (Table 29). The ±12-line tolerance stands in for the hunk that the fix commit's changed lines sit in rather than for a program element: it is coarser than the statement granularity spectrum-based work scores and finer than a file, and it exists because the coordinates the model cites in are not the coordinates the fix was recorded in (Appendix G.4), not because a line is the unit we think matters.

Mechanism naming scored from inputs other than a commit. Four of the works the main text cites do name a cause, and each names it from evidence we do not have. TorchTraceAP (Chen et al., 2025a) releases over 600 PyTorch traces from computer-vision models collected on several hardware platforms and scores the detection of performance anti-patterns inside trace segments, with a small classifier proposing segments and a language model classifying them: the input is a profile of a run and the unit is a segment of that profile, so it has no inducing commit and no fix site to score. Sijwali et al. (2025) fine-tune a small model to detect, repair and explain performance bugs in Java from 392 labelled bugs across 17 Defects4J projects, with code diffs, developer comments and bug reports as input, and grade the explanation against the developer's own report; the scored object is free text matched to a report, not a label from a fixed set, and the corpus is a Java functional-bug benchmark. Li & Wang (2026) is the closest in population and in method cost: it annotates issue reports and developer discussions for LLM inference engines with a root cause category and the affected module, classifies both from text with nothing executed, and evaluates on vLLM issues with a transfer check to TensorRT-LLM. Its unit is the issue report where ours is the commit, and its taxonomy is built for issue text and carries no performance mechanism class of the kind we label, and its localisation is a module where ours is a window of 25 lines. Ekka (Gu et al., 2026) diagnoses silent output-quality errors in serving frameworks by aligning intermediate execution states against a reference implementation known to be correct. That construction does not transfer to us: a performance regression has no correct reference run to diff against, which is why our oracle has to be the commit that repaired it. IOAgent (Egersdoerfer et al., 2025), which the main text cites for the priority point, diagnoses HPC I/O performance from a Darshan trace with retrieval over domain documents, and it too takes a trace of an execution as its input.

Execution-based repair and optimisation benchmarks. SWE-bench (Jimenez et al., 2024) established the fix-commit-as-oracle construction for functional bugs, where a failing test makes the oracle checkable by execution. The performance analogues (He et al., 2025; Shetty et al., 2025; Ma et al., 2025) inherit the construction and pay the execution cost, so their items are limited to regressions whose workload their authors could run. Ours is the complementary population, and we drew it at the repository level: every item comes from one of the four distributed training and inference repositories of §2, where the workload that exposes a regression needs hardware and a harness the repository does not ship, so the diagnostic question has to be asked without execution or not at all. We did not test the item-level version of that claim: no predicate in `code/census.py` or in the eligibility funnel (Appendix A) asks whether a regression reproduces on one device, so nothing here says that each of the 430 items needs the multi-GPU configuration it came from. ISO-Bench

(Nangia et al., 2026) is the fourth of that group and the closest to our repositories: 54 optimisation tasks taken from vLLM and SGLang, scored against expert human patches by execution-based and model-based metrics together, with the agent handed a description of the bottleneck along with the codebase. Diagnosis is its input. SWE-Perf hands over the target functions as part of the instance, while GSO and SWE-fficiency leave the localisation to the agent and score the patch it writes, not what it concluded, so none of the four scores a diagnosis. ISO-Bench's own result is the argument for scoring one: agents there often identify the correct bottleneck and still fail to produce a working patch. ISO-Bench also reports a result that runs against our second contribution, and it needs an answer here: agents that share an underlying model differ from each other on its tasks, and its abstract reads that as scaffolding being as important as the model. Both results can hold, because the two sweeps vary different objects. ISO-Bench varies the agent harness on a task whose output is a patch and whose environment can be executed, so the harness decides how many edits, test runs and retries a model gets. Our sweep varies the content of one prompt, on one model, for one diagnosis call, with nothing executed and no second sample. Ours is therefore evidence about prompt content and not about the harness around the call, and the two harness families ours leaves untested are search over a tree of decisions with feedback (Yao et al., 2023) and multi-agent division of labour (Du et al., 2023). The strong reading of C2, that no scaffold helps on this task, does not follow from our data and §4 does not assert it.

Bug-dataset construction from version control. Defects4J (Just et al., 2014) and Denchmark (Kim et al., 2021) are the reference points for mining (buggy, fixed) pairs, the latter in deep-learning software with line-level localisation derived from fix commits. TreeVul (Pan et al., 2023) is the closest methodological precedent for our label side: commit diffs mapped onto a fixed hierarchical taxonomy at larger scale than ours, for vulnerabilities. What we add over both is the population, AI-infrastructure performance regressions rather than functional bugs or vulnerabilities, and a measured label reproducibility. We did not find a label-reproducibility measurement in either: neither claims one, and we are asserting only what we looked for and did not find, since both label sets are large enough that an unreported internal check is possible. It is the measurement we consider necessary once labels are not human-authored.

Training-data contamination controls. Our *Diff withheld* column (§6 calls this the closed-book condition) bounds what a model can do without the diff, and §6 is explicit that this is a bound and not a membership claim. We stop there because deciding membership properly is its own research problem. A membership test here would have to work from model outputs alone, since we do not hold the pre-training corpora, and per-record tests of that shape cost many queries per candidate. That budget is not available to us for 430 commits, and the verdict such a test would return on four public repositories is close to a foregone conclusion: public code is the part of the pre-training corpus a positive result is least surprising on, so the information per query is low and the cost per query is not. We therefore do not claim these items were unseen; we bound what the diff adds, and that bound holds either way.

There is also a reason a membership verdict would not settle the question even if we could obtain one. Chen et al. (2024) show that what a model will emit about its training data is not fixed by pre-training: fine-tuning can surface content the base model does not produce. The checkpoints we score are post-trained, and we do not control or observe that stage, so a clean verdict on a base model would not transfer to them, and a positive one would not tell us whether the item's mechanism, as opposed to its text, is what the model recalls. This is why our control is a closed-book *arm* rather than a contamination filter: it measures what the model can do when the diff is withheld, on the same checkpoints and through the same harness as every other number here. We report that gap and do not claim to know which items were memorised.

Models as annotators and as measurement instruments. Every label here was authored by a model, and §2 reports the two-pass agreement that bounds what such labels can support. The substitution itself is not novel, and the risk it carries here is documented: language-model judges prefer their own outputs (Wataoka et al., 2024; Chen et al., 2025c), and the effect persists when the judgement is made against an explicit rubric rather than free-form (Pombal et al., 2026), which is exactly the shape of our label set. Three of the 6 evaluated models are also panel annotators, so a model can be credited for agreeing with a label it helped write. We cannot remove this, so we bound it. Appendix J states a self-preference criterion we committed before reading any of these labels,

then re-scores every model against a cause-family key built only from the other annotators' votes; that key leaves the Table 1 order unchanged (Table 41). Table 38 prints every subset whose key Claude Opus 5 did not write alone, including the cross-vendor key swap over all 60 re-labelled items, which selects none of them, and the 49-item subset the cross-vendor annotator confirmed, which selects easy ones. Appendix E splits every model by label source, and the panel column is where a reader should look for the size of the reuse. What we do not have is a human-authored key. One case is sharper than label reuse. The symptom class the *Diff withheld* prompt shows is the annotation card's own field: it equals that field on 622 of 622 pairs and the imported upstream prior on only 380, so it was written by the annotation pass and not observed independently. On 324 of the 344 scored items that pass also wrote the gold label, and that pass is Claude Opus 5, so for that model the closed-book cell is in part a read-back of its own annotation. On the other 20 the card carries no label and the gold came from the panel, and 12 of those cards come from the deep-read pass, whose cards carry no label field at all rather than an empty one, so the read-back reading does not extend to them. None of this removes the need for a human expert pass. We have not run one, and §8 does not ask for one: it opens on the correctness bound split by provenance, and lists the 19-item author check, which was neither blind nor given a planted error to catch, among three checks that are weaker than they look.

Scaffolds this sweep does not cover. The 20 variants change prompt content, label presentation, evidence access, sampling and graph depth. They leave two broader approaches untested. Tree search with feedback between branches (Yao et al., 2023) permits revisiting decisions, whereas our graph walk commits to one branch at each layer. Our measurements do not isolate routing from graph structure, so a method that revises a mistaken route (Madaan et al., 2023) could behave differently. Multi-agent division of labour (Du et al., 2023) is also untested: every evaluated arm uses a single model.

The supported result is that none of the tested prompt variants yields a statistically resolved improvement for Claude Sonnet 4.5 on 67 dev items. Given the sweep's limited resolution, this does not establish that prompting is unimportant or that other scaffolds cannot help. The differences among 6 models in Table 1 provide separate evidence and do not extend the single-model prompt result.

Performance-bug taxonomies. The hand-written line is the older one and the one our label set has to answer to. Jin et al. (2012) categorise 109 performance bugs from server, browser, compiler and database code and name 3 root causes; Selakovic & Pradel (2016) name 8 over 98 fixed issues in 16 JavaScript projects; Zaman et al. (2012) read 400 bug reports and Nistor et al. (2013) compare how performance and non-performance bugs are discovered, reported and fixed, so those two describe the practice around such bugs and publish no root-cause category set. Appendix I maps our 32 leaves onto the 11 categories the first two do publish, in both directions. Li et al. (2023) is the closest of this line to our population: a categorisation of performance bugs in distributed storage and computing systems, built for empirical analysis and static detection rather than for scoring a model; it publishes no model baselines and no held-out split, which is a difference in purpose and not a defect. Our 32-label set is narrower by construction, covering only mechanisms that appear in AI-infrastructure regressions, and is released with the floors, two-pass agreement and splits that make it usable as an evaluation target rather than as a classification scheme.

N What the benchmark measures

Statistical resolution of the prompt sweep. The null result in §4 must be interpreted at the resolution of the paired tests. `code/mde.py` asks, for every (variant, metric) cell with a paired dev test, how large a difference the two-sided exact McNemar test would have rejected at $\alpha = 0.05$ *given the number of items that actually disagree*. Across 40 cells the answer is 10.2 to 17.9 accuracy points, median 13.5. On 2 cells the answer is that no difference is detectable at all: with five or fewer disagreeing items, every split of them has $p > 0.05$, so those cells carry no information about whether the variant helps and should not be read alongside the others as evidence that it does not. A scaffold worth ten points would therefore have gone unnoticed in most of this sweep. The result in §4 therefore concerns detectable improvements under this design; it does not exclude smaller gains. `results/mde.json` carries the per-cell numbers.

Joint mechanism and fix-site accuracy. The *Both* column of Table 1 requires the exact mechanism *and* a fix site within 12 lines, on the same item. It runs from 0.080 to 0.376. For the strongest model that joint rate is 0.72 of its exact-mechanism rate and 0.66 of its fix-site rate: the two are correlated on an item but far from implied by each other. Joint accuracy is relevant when a diagnosis must identify both the mechanism and the code to change. Its practical interpretation still depends on an untested assumption that agreement with the key predicts usefulness: we do not evaluate patches written from predicted mechanisms or obtain engineer ratings of their usefulness (§8).

Discriminator format and correctness. Models return a confirm/refute pair the scorer reads as well formed on 0.974 to 1.000 of items. Whether the pair actually *discriminates* is not measurable here: checking it means running the code the design avoids. So we report well-formedness only and keep the discriminator out of the joint *Both* column. Evaluating correctness would require additional evidence. Using a model judge would introduce another model-dependent component to the evaluation.

Examples of discriminator outputs. The rate above counts an answer as well formed when `confirm_prediction` and `refute_prediction` both come back non-empty under those names (`code/score.py`, `score_m3`). Whether either side is falsifiable is not tested by anything, so the rate is worth no more than the pairs behind it, and this paper has quoted none. The two below are copied from the ladder runs' per-item files, `probe/runs/mt32k__main430__model__split__rs3.jsonl`. GPT-5.6's answer to the Figure 2 item, `vllm@26b5f7bd2a40` repaired by `031ca762d7bd`, carries the label and the fix site card 2 prints, and its pair reads: confirm, "Statically, for at least one supported non-CUDA platform, current_platform.is_cuda_alike() is true while current_platform.is_cuda() is false; on that platform this file no longer binds the four fusion-pass symbols, whereas it did before this commit."; refute, "If is_cuda_alike() and is_cuda() have identical truth values on every platform that can reach this pass manager, or every non-CUDA CUDA-like pass-manager path neither registers nor attempts to use any of these four passes, this predicate change cannot explain a regression." Both sides name a fact in the tree a reader can check without running anything, which can in principle be checked within the available information. For the negative, Llama-4-Maverick on `DeepSpeed@351569dd4a00` stated one side, "The code will correctly handle switching between training and evaluation modes.", and wrote the other under the key `refute` instead of `refute_prediction`, so the scorer saw no second side and the item counts against the rate. The side it did state would not have carried the pair anyway, because it names no path and no work and so cannot fail. A second failure mode in these files is a transcription artifact and not a missing thought: Haiku 4.5 on `Megatron-LM@de56227b26ba`, in its dev file, wrote its refute side inside the confirm string as the text of a tool-call wrapper, so the field the scorer reads is empty while the sentence is there to read.

Item-level agreement across models. Of the 344 items, only 36 are named correctly at the family level by all six models and 77 by none; 231 split them. A predominance of trivially easy or uniformly difficult items would provide less information about differences between models. The 77 items missed by every model account for a concentrated part of the remaining error (Appendix E).

O Limitations, licensing and misuse

Data provenance and licence. All items name commits in public repositories: DeepSpeed, TransformerEngine and vLLM under Apache-2.0 and Megatron-LM under BSD-3-Clause, with third-party files in Megatron-LM carrying their own notices, each read at the commit the corpus lock file pins (`code/licences.py` writes `results/licences.json`, and both this sentence and the Ethics statement print it from there, so a changed pin cannot leave one of them behind). We release commit identifiers, labels and splits, not repository contents, so the release carries no upstream code and inherits no upstream licence obligation beyond attribution. The release itself is licensed: the corpus records, labels and splits under CC BY 4.0 and the harness, scorer and generators under the MIT licence, both stated in the `LICENSE` file the archive carries. No item contains personal data beyond what a public git history already exposes; we do not release commit author names or emails, and the harness does not put them in prompts.

Scope of the conclusions. Four repositories, Python-dominant, inference and training

Missing control for family information and menu length. No arm states the gold cause family inside the flat 32-label menu, so the 2.00× and 1.55 × contrasts of Table 17 move the family label and the menu length together. The decomposition there bounds the menu on the correctly-routed half only, on 3 and 6 discordant items, where no split of its 3 discordant items reaches 0.05, so this cell carries no resolution at all and the smallest difference it could have rejected is 6 of its 39 items, 0.154 accuracy; on the misrouted half, which supplies the whole gain, the two are collinear by construction and no arm can tell them apart. The arm that would is a flat call that states the gold family and keeps all 32 labels, on the 67 dev items for the two models §5 sweeps. We did not run it, and `code/runner.py` implements `family_oracle` only inside the graph arm, so every statement in this paper about what the family label is worth is an upper bound.

infrastructure. Nothing here transfers to compiled kernels, CUDA source, or other languages. Six models from 3 vendors at one point in time; §5's finding that the graph's damage is *not* monotone in capability is a statement about these six and no others. The graph is also one design: Claude Opus 5 wrote all of its nodes under prompt `node_author.v1`, and that model also wrote the answer key for 324 of the 344 scored items. Every graph cell in this paper walks that one graph, so §5 bounds this scaffold and not hierarchical diagnostic scaffolds in general. We never ran a graph from another source, hand-written or authored by any other model.

The mechanism rates are scoped to one naming convention. Every mechanism number here is strict single-leaf match against a key whose own annotators accept a substitute mechanism on 0.903 of items once the family they chose leaves the menu (Appendix I). Read a mechanism rate as accuracy under this taxonomy's naming and not as a level of diagnostic skill: the redundancy-aware variants of Table 35 keep the model ordering under 3 of 4 looser conventions but shrink best over worst from 2.95 to 2.00, and a user who renames or merges leaves will move the absolute rates without touching a single diagnosis.

Limits of two-pass agreement. A model at 0.794 family accuracy matches our key as often as a second pass does. It has not been shown to diagnose performance regressions correctly, only to agree with a second machine-authored pass as often as that pass agrees with the first.

The oracle is a single fix commit. When a regression was repaired in more than one place, or refixed later, our answer key credits the first repair only. The fix-site metric inherits this and will call a correct alternative site wrong; the mechanism label inherits it too, in the weaker sense that a regression with two mechanisms is filed under whichever one the first fix addressed. We did not find a way to detect the multi-fix case mechanically without executing anything, so we do not know how many items it affects.

Fix-site scores relative to a simple heuristic. Citing the three largest hunks of the inducing diff scores 0.328 at line level; the weakest model we ran ties it (0.305, exact McNemar $p = 0.38$), though the heuristic cites three sites to that model's 1.40 and at one citation it scores 0.206, which the model clears (p <0.001); four of the six sit between 0.407 and 0.451 and separate from it in the paired test, and only 0.570 is well clear of the floor. Treat M2 as a sanity check on M1 rather than as an independent capability measurement, and quote it only with the floor attached.

Misuse. The most plausible misuse is treating a high score as licence to deploy an automated regression triager. The decoy results forbid this: under neutral framing these models claim a regression on 0.129 to 0.549 of commits that contain none across the 9 (model, stratum) cells, and the spread is wide enough that the rate has to be measured per model and per stratum before any deployment argument is made. A second misuse is using the fix-site column as evidence of code-localisation ability, which the 0.328 heuristic floor and the 1.995× naming-over-pointing asymmetry of §4.1 both argue against.

Contamination. Fix commits for most items predate the training cutoffs of the models we evaluate, and we cannot rule out that a model has seen the fix. The closed-book control bounds what the diff adds, not what was memorised: with the diff removed the strongest model reaches 0.340, and the symptom-only zero-call lookup table of Appendix G.5 reaches 0.331 [0.284, 0.383] on those 344 items. That table matches the stronger cell (discordant 63:60, exact McNemar $p = 0.86$) and exceeds the weaker one, 0.262 ($p = 0.038$, which does not survive our BH correction); it is not above both

cells. So the closed-book *level* carries no evidence of a memorisation signal at family granularity and only the gap to the open-book cell does. At mechanism granularity it does carry one: 0.174 against 0.087 (p <0.001) when the 49 items whose symptom is `n/a` fall back to the global modal family, and against 0.096 (p = 0.0027) when `n/a` is treated as a class, which is what the prompt prints, both on the 311 items with a leaf-level gold label; 0.106 is separated from neither (p = 0.47 and p = 0.78). We give both treatments because the lower floor is the one that makes this comparison, the only one here that favours the closed-book arm, look largest. For a stricter test the released items carry fix dates, so a subset after any given cutoff can be selected, but not from the pool we score: 0 of the 344 scored items have a fix date after 2026-05-01, the cutoff this benchmark uses to define its own freshest window, and the newest scored fix is 2026-03-17. The 50 items that do fall after it are all in the sealed test split, so the stricter test this objection asks for is unavailable until that split is opened. Withholding the symptom line as well is the control this objection really asks for, and for Opus 5 it removes about the same share of what is left as withholding the diff removed, a smaller absolute drop: on the same 344 items Opus 5 falls to 0.198 [0.159, 0.243], so the symptom line alone is worth 0.142 of its closed-book cell against the 0.244 that withholding the diff cost (p <0.001), and what is left is not separated from the 0.166 majority-class floor (p = 0.17), while sitting well below the symptom-only table (p <0.001). Sonnet 4.5 ran on the same items and also falls, to 0.128 [0.097, 0.167]: there the symptom line is worth 0.134 (p <0.001) against the 0.108 the diff is worth, an ordering reversed from Opus 5's, and what is left is not separated from the 0.166 floor (p = 0.11) while sitting below the symptom-only table (p <0.001). Both models' runs are the `cbs430` rows of `results/tables/app_allruns.tex`. That arm also rewords the instruction sentence, so it is not a one-factor contrast.