RETHINKING CODE LOCALIZATION THROUGH REPAIR-CONTEXT COMPLETENESS: FROM STRUCTURAL AND CODE COVERAGE ANALYSIS TO CONTEXT AUGMENTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Repository-level code localization is difficult because a single repair may span multiple distant source regions. Existing evaluations often report relevant hits or file coverage, but these metrics alone are insufficient to determine whether a bounded repair context contains all regions involved in a fix. We characterize this gap through repair-context completeness, defined as containment of the full *repair closure*, which consists of the pre-fix regions changed by a reference patch. Within this scope, we examine the assumption that directly using execution coverage improves line-level localization. Experiments across six dimensions show that execution evidence can expose more candidates without reliably assembling complete repair contexts under the tested policies. Accordingly, we introduce Complete-Fix Reachability–Realization (CFRR), a measurement framework that attributes incomplete contexts to failures at distinct stages of localization and context construction. Guided by this diagnosis, we propose TRACEFILL, which preserves the semantic localization base and uses repeat-consistent execution coverage to fill remaining context capacity. Our evaluation spans three models and starts with 1,100 instances from three datasets. TRACEFILL increases full repair-closure containment from 33.38% to 41.24% compared with the semantic baseline, a gain of 7.86 percentage points. In the downstream evaluation, repair success increases over Native for all three repair generators, with absolute gains of 0.60–1.69 percentage points and relative gains of 23.5–32.7%.

1 INTRODUCTION

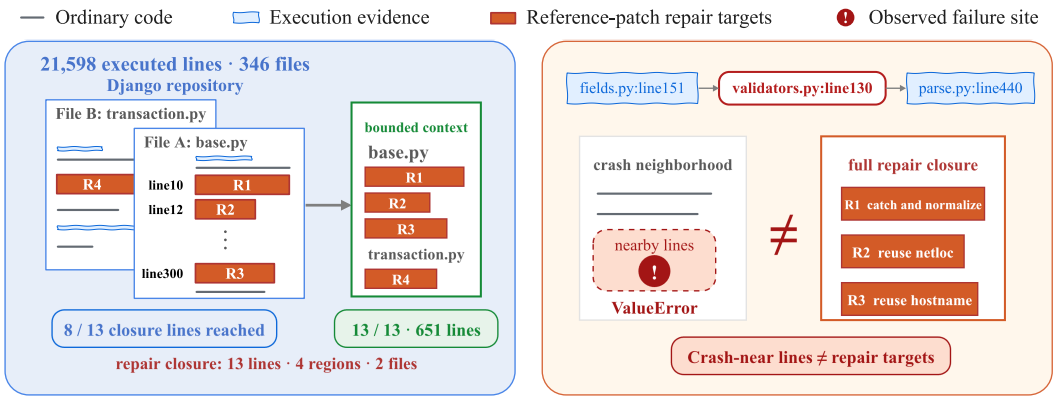
Repository-scale repair may require coordinated changes across distant source regions (Jimenez et al., 2024; Xin et al., 2024). Finding a relevant file or line does not establish that the context supplied to a repair model contains all regions involved in a reference repair. We study *repair-context completeness*: whether a bounded source context contains the full set of pre-fix regions mapped from a reference patch, termed its *repair closure*.

Hierarchical and graph-based localization narrow the search space (Xia et al., 2025; Chen et al., 2025), while coverage, dynamic slices, and crash stacks provide evidence from failing-test execution (Agrawal & Horgan, 1990; Jones et al., 2002; Wu et al., 2014; Wong et al., 2016). However, broad coverage can exceed the available context budget, and crash proximity can omit distant repair regions (Figure 1). The challenge is to turn this evidence into a complete context within a source-line budget.

In our development cohort, completeness failures persist even when the repair file is available. Selection experiments further show that broader candidate pools need not yield more complete contexts under the tested policies. We introduce Complete-Fix Reachability–Realization (CFRR) to distinguish file exposure, candidate reachability, budget feasibility, and the completeness of the selected context.

This distinction motivates evaluating execution evidence as a complement to a semantic localization base. We propose TRACEFILL, which preserves the frozen Native context and adds regions from

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.



(a) Few target lines across many files **(b) Crash proximity ≠ repair closure**

Figure 1: Two limitations of execution-guided localization in Django. **(a)** Broad execution coverage does not ensure that a bounded repair context contains repair regions across files and distant locations. **(b)** A traceback identifies the crash site, but nearby code may omit the full repair closure. Line numbers in (a) are schematic.

repeat-consistent failing-test execution within residual source-line capacity. Additions are selected without reference-patch information. We evaluate repair-closure containment and assess downstream repair separately.

Our contributions are threefold:

- We formalize reference-patch-based repair-context completeness. Our development studies show that file exposure and broader candidate pools do not guarantee complete contexts under the tested selection policies.
- We introduce CFRR, a four-stage diagnostic framework that assigns each incomplete context to its first unmet stage: exposure, reachability, capacity, or realization.
- We propose TRACEFILL, a Native-preserving, gold-blind context completion policy. Across three models and 763 defects, it raises repair-closure containment from 33.38% for Native to 41.24% with up to 414 additional source lines.

2 RELATED WORK

Set-completeness objectives appear in localization and repair systems. Agentless reports Contains-GT and average LoC for edit-location windows (Xia et al., 2025); SWERank's Acc@k requires all relevant locations, while Repository Memory and HitMore measure full reference-file coverage (Reddy et al., 2026; Wang et al., 2026; Xu et al., 2025). Our contract instead scores the full reference-mapped repair closure under a hard source-line limit. SWERank and Repository Memory also evaluate downstream repair, while ITER accumulates multi-location edits through repeated localization, generation, and validation (Ye & Monperrus, 2024).

MANIPLE selects static, dynamic, and external facts under function-granular perfect localization (Parasaram et al., 2025), while execution-trace prompting inserts runtime traces into generation prompts (Haque et al., 2025). Our matched addition controls assess context selection within the frozen pipeline; they do not establish TRACEFILL's relative performance against established localization or repair systems under matched budgets. Appendix A compares the mechanisms and evaluation contracts.

3 REPAIR-CONTEXT STRUCTURE AND EXECUTION EVIDENCE

RQ1 relates file availability and repair structure to context completeness. RQ2 then tests execution coverage and crash proximity as signals for selecting complete contexts. RQ3 refines this analysis

with CFRR, distinguishing failures of file exposure, candidate reachability, budget feasibility, and realization. RQ4 builds on this diagnosis to evaluate Native-preserving completion with TRACEFILL.

3.1 STRUCTURAL CHARACTERISTICS OF REPAIR CONTEXTS

The structural analysis considers both access to the repair file and the distribution of edit locations within it. The cohort contains 117 development defects with single-file, on-path reference patches. For defect i, N_i is the union of anchors from four retained Native edit-location samples; Section 5 instead fixes one Native base per defect and model. The nonempty set G_i contains the pre-fix reference-patch lines retained by the declared mapping and cleaning rules. Localization outputs are frozen before reference scoring (Appendix C). Each anchor receives one clipped ± 10-line expansion, and $W_i(N_i)$ is the rendered union with overlaps counted once. Native is complete when this union contains G_i; for cohort $\mathcal{I}$,

$$C_i^{\text{Native}} = \mathbf{1}[G_i \subseteq W_i(N_i)], \qquad \text{NativeComplete} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} C_i^{\text{Native}}. \tag{1}$$

Native is complete for 77/117 defects. Given the cohort's single-file, on-path scope, these structural findings and CFRR frequencies do not establish cross-file failure distributions. Table 1 describes five overlapping conditions covering repair-file availability and the line, hunk, function, and region counts. A region is a maximal run of consecutive reference-mapped source lines in one file, counted before expansion.

Table 1: Structural composition of the 117-defect development cohort. Entries count defects; conditions overlap.

Condition	With condition	Without condition
Repair file is available	104	13
Reference patch changes > 1 line	96	21
Reference patch has > 1 hunk	43	74
Changed lines span > 1 function	33	84
Changed lines form > 1 region	54	63

All 117 defects are single-file and on the crash path; these are fixed cohort properties.

Repair-file omission accounts for 13/40 incomplete Native cases; 27 remain incomplete despite file availability. Completeness is also lower for distributed edits: Native completes 28/54 multi-region versus 49/63 single-region defects, and 17/43 multi-hunk defects versus 60/74 in their complement (Appendix F). These overlapping, full-cohort comparisons are descriptive and do not isolate fragmentation from file availability or repository effects. Together, the file-availability gap and structural contrasts motivate treating context composition as a joint selection task: the rendered context must cover all mapped repair regions within its source-line allowance. The next section examines execution evidence for supplying and selecting these regions.

3.2 EXECUTION EVIDENCE FOR CONTEXT SELECTION

Six studies examine whether and how execution evidence can support complete-context construction under a source-line budget (Figure 2). They assess candidate supply and selection, closure overlap, behavioral response, and directional certification as distinct outcomes. Experiments 4–6 are exploratory probes on small development subsets and do not establish population-wide bounds. Pools, rankings, and contexts are frozen before reference scoring. Here, *Complete* scores one context; Section 5 defines three-budget COMPLETE@3. Appendix G details the protocols and statistical analyses.

(1) Crash proximity often adds no new context. Across 101 prespecified baseline-incomplete defects, the crash-proximity policy yields a mean completeness difference of $+0.008$ percentage points relative to Native-preserving controls with matched added-anchor counts and comparable incremental source-line costs. This falls below the prespecified $+5$-point practical threshold. A post-hoc decomposition finds that 79 actions add no source lines beyond Native. The remaining 22 average $+0.036$ points, with inconclusive selection efficacy. The overall result therefore cannot establish how well crash proximity selects code when additions are available.

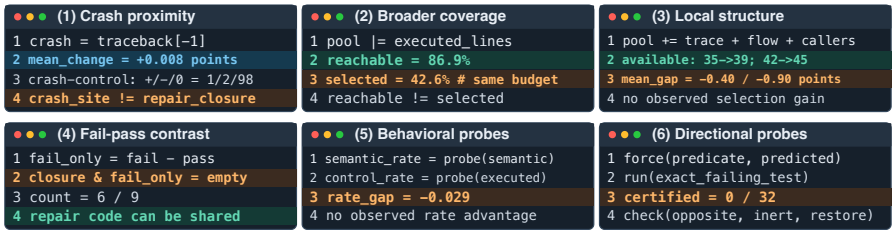


Figure 2: Six policy-specific studies of execution-derived signals for complete-context selection, covering candidate supply, selection, behavioral response, and directional certification under their respective contracts.

(2) Broader candidates do not ensure better selection. On 61 development defects, an expanded pool including whole-module proxy coverage has an 86.9% gold-aware closure ceiling. Yet, at a common cap equal to Native's mean rendered size, completeness is 42.6%, compared with 65.6% when the same selector uses only Native anchors. Because selection starts empty, additions compete with Native for capacity. This policy-specific regression motivates preserving the semantic base during completion.

(3) Local relations expand closure availability. More structured expansion also leaves a gap between availability and selection. Crash-stack, definition–use, and caller relations increase closure availability from 35 to 39 defects in development cohort 1 and from 42 to 45 in cohort 2. At 90% of Native's rendered size, however, completeness is 60.71% versus 61.11% for matched randomized controls in cohort 1, and 60.66% versus 61.56% in cohort 2. The additional reachability therefore does not translate into an observed advantage over matched random additions (Appendix G.3).

(4) Failure-exclusive execution can omit the repair closure. In six of nine development defects, the repair closure has no overlap with lines covered by the failing test but not the paired passing test. Reference-mapped repair lines may also be covered by passing tests or lie outside the failing trace. Thus, restricting selection to failing-only execution can discard repair-relevant code. This limits passing-test subtraction as a direct selection rule without ruling out other uses of coverage.

(5) Behavioral influence at semantic candidates. Reversible edits reveal no detected response advantage for semantic candidates over matched executed sites. Across 17 paired development defects, the mean within-defect difference in informative-response rates is -2.9 percentage points, with permutation $p = 0.778$. Informative responses change the original failure without introducing an earlier exception, or make the exact failing test pass. These responses measure behavioral influence, while the groups are defined by candidate provenance; the comparison does not establish repair-closure membership.

(6) No tested direction achieves full certification. Testing a predicted repair direction requires evidence beyond an informative response. Certification requires the predicted direction to repeatedly resolve the failure while the target executes, the opposite direction to preserve the original failure, and a behavior-preserving control to remain inert. None of 32 predicted Boolean directions across 17 defects satisfies the full certification protocol (Appendix G.6).

Together, these studies motivate evaluating execution evidence through its contribution to complete-context construction. The observed gap between candidate availability and complete selection calls for a diagnosis of where construction falls short. CFRR provides this stage-wise analysis by separating file exposure, candidate reachability, budget feasibility, and realization.

4 CFRR: A Stage-Wise Analysis of Repair-Context Construction

4.1 Stage-Wise Diagnosis and Failure Attribution

A binary completeness score shows whether context construction succeeds, but not where it fails. CFRR turns this outcome into a diagnosis by checking the nested requirements for a complete context.

Assigning each failure to the first unmet requirement yields mutually exclusive attributions and identifies which part of context construction warrants further examination.

The four stages test progressively stronger conditions: ① Exposure asks whether all repair files are available; ② Reachability, whether frozen candidates contain a complete context; ③ Capacity, whether one fits the source-line limit; and ④ Realization, whether the gold-blind policy returns one. For defect i, $\mathcal{U}_i$ is the source in upstream-localized files. The action family $\mathcal{A}_i$ contains all subsets of frozen candidate anchors within these files, including Native's anchor union N_i (Appendix H). For the Native audit, $\Pi_i = \{N_i\}$ and $B_i = |W_i(N_i)|$. The repair closure G_i and rendering operator W_i follow Section 3.1, including window expansion and overlap removal. With Fit_i denoting Capacity, the indicators are

$$\begin{aligned} \mathrm{Exp}_i &= \mathbf{1}[\mathrm{files}(G_i) \subseteq \mathrm{files}(\mathcal{U}_i)]\,, \\ \mathrm{Reach}_i &= \mathbf{1}[\exists A \in \mathcal{A}_i : G_i \subseteq W_i(A)]\,, \\ \mathrm{Fit}_i &= \mathbf{1}[\exists A \in \mathcal{A}_i : G_i \subseteq W_i(A) \ \wedge\ |W_i(A)| \le B_i]\,, \\ \mathrm{Real}_i &= \mathbf{1}[\exists A \in \Pi_i : G_i \subseteq W_i(A) \ \wedge\ |W_i(A)| \le B_i]\,. \end{aligned} \tag{2}$$

Native automatically satisfies its budget, so $\mathrm{Real}_i = \mathbf{1}[G_i \subseteq W_i(N_i)]$. Because Native belongs to $\mathcal{A}_i$ and all actions draw from $\mathcal{U}_i$, $\mathrm{Real}_i \le \mathrm{Fit}_i \le \mathrm{Reach}_i \le \mathrm{Exp}_i$: a realized complete context must fit, a fitting context must be reachable, and a reachable closure must lie in exposed files. Successive pass-rate differences therefore partition incomplete outputs by their first failed condition (Figure 3). These attributions identify the first unmet requirement under fixed inputs, without establishing independent causal effects of changing individual stages.

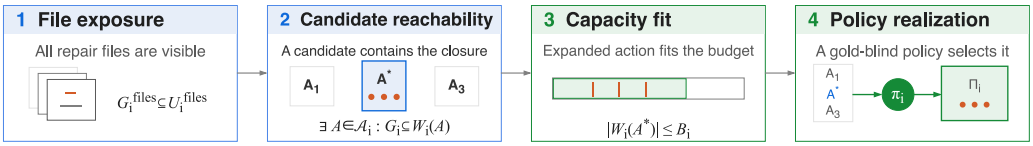


Figure 3: CFRR separates Exposure, Reachability, Capacity, and Realization failures. The first three are gold-aware diagnostics over frozen inputs; Realization evaluates Native's frozen gold-blind output.

4.2 CFRR Analysis Results

Realization is the largest first-failure category in the 117-defect single-file, on-path development cohort, accounting for 26 of Native's 40 incomplete contexts (Table 2). Of the 103 defects admitting a complete candidate within Native's source-line budget, Native completes 77. This gap persists despite file exposure, candidate reachability, and capacity fit. The audit thus separates the existence of a feasible complete context from its realization by the evaluated gold-blind output.

Table 2: CFRR audit of 117 single-file, on-path development defects. Capacity tests oracle subset feasibility without Native preservation. Pass rates are cumulative; conditional rates use preceding-stage survivors; first failures are mutually exclusive.

Stage	Passed stage	Conditional rate	First failure
Exposure	104/117 (88.89%)	104/117 (88.89%)	13/117 (11.11%)
Reachability	103/117 (88.03%)	103/104 (99.04%)	1/117 (0.85%)
Capacity	103/117 (88.03%)	103/103 (100.00%)	0/117 (0.00%)
Realization	**77/117 (65.81%)**	**77/103 (74.76%)**	**26/117 (22.22%)**

Completing an incomplete context while preserving Native requires explicit residual capacity. The zero Capacity-failure count establishes that gold-aware subset selection fits all 103 reachable closures within Native-sized budgets, but this diagnostic allows replacing Native content. Under strict KEEP, an action retains N_i and therefore all source lines in $W_i(N_i)$. If it must also fit within $|W_i(N_i)|$, its rendered context must equal Native's. Zero-cost anchors cannot supply missing source lines. The original capacity result therefore does not extend to Native-preserving completion. This distinction motivates TRACEFILL's explicit residual allowance, evaluated separately in Section 5 with one frozen Native base per defect and model.

5 TRACEFILL: GOLD-BLIND REPAIR-CONTEXT COMPLETION

5.1 TRACEFILL

TRACEFILL addresses the residual-capacity requirement identified in Section 4 by preserving Native and adding execution-derived source regions under a fixed priority order. Native anchors are frozen once per defect–model pair and reused across all arms and budgets. Parsing failures uniformly yield an empty base, which does not imply an empty model response (Appendix N.2).

For defect i and model $m \in \mathcal{M}$, the localizer supplies Native anchors S_{im}, preserved as KEEP, $K_{im} := S_{im}$. All arms use residual caps $b \in \mathcal{B} = \{51, 170, 414\}$, measured in additional rendered source lines beyond Native. Additions Δ_{imb} satisfy the total rendered limit $B_{imb} = |W_i(S_{im})| + b$:

$$\widehat{S}_{imb} = S_{im} \cup \Delta_{imb}, \qquad |W_i(\widehat{S}_{imb})| \le B_{imb}. \tag{3}$$

TRACEFILL draws its candidates from repeated execution of the declared failing-test input. Let $E_i^{(1)}$ and $E_i^{(2)}$ be the valid Python line sets from two qualifying runs with the same exception. Their intersection retains lines observed in both runs:

$$T_i = E_i^{(1)} \cap E_i^{(2)}. \tag{4}$$

Candidates outside KEEP are ordered by provenance: repeat-consistent exception sites, call/return endpoints, then other executed lines, with ties resolved by Native file rank, file path, and line number. Each anchor receives one clipped ± 10-line expansion. Because windows overlap, its marginal cost counts only source lines not already selected; equal anchor counts can therefore consume different amounts of capacity. Starting from KEEP, TRACEFILL accepts a window if this marginal footprint fits and skips it otherwise, continuing to consider later windows. At larger caps, the same frozen order is rescanned while retaining accepted anchors, allowing previously skipped candidates to use the expanded allowance. The ordering allocates residual capacity among available candidates without using the reference patch to choose additions.

Completeness evaluates whether the resulting context contains the entire reference-mapped closure. For method A, let A_{imb} be its output anchors and $\mathcal{P}$ the eligible model–defect pairs, fixed across caps by the declared availability and matching rules. Candidates, priorities, caps, and outputs are frozen before reference scoring. Define

$$\begin{aligned} C_{imb}(A) &= \mathbf{1}[G_i \subseteq W_i(A_{imb})], \qquad \mathrm{Complete}_{\mathcal{P}}(b; A) = \frac{1}{|\mathcal{P}|} \sum_{(i,m)\in\mathcal{P}} C_{imb}(A), \\ C_{im}^{@3}(A) &= \max_{b\in\mathcal{B}} C_{imb}(A), \qquad \mathrm{Complete@3}_{\mathcal{P}}(A) = \frac{1}{|\mathcal{P}|} \sum_{(i,m)\in\mathcal{P}} C_{im}^{@3}(A). \end{aligned} \tag{5}$$

The balanced primary panel has $\mathcal{P} = \mathcal{I} \times \mathcal{M}$; supporting comparisons use their stated pair sets. For each pair, COMPLETE@3 indicates whether any prescribed budget output contains the closure. Native's fixed context and TRACEFILL's nested outputs make COMPLETE@3 equal to Complete(414) for these two methods.

The construction in Figure 4 preserves Native while respecting the source-line limit. For every eligible defect–model cell, the returned anchors $\widehat{S}_{imb}^{\mathrm{TF}}$ satisfy

$$K_{im} \subseteq \widehat{S}_{imb}^{\mathrm{TF}}, \qquad |W_i(\widehat{S}_{imb}^{\mathrm{TF}})| \le B_{imb}, \qquad C_{imb}^{\mathrm{TF}} \ge C_{imb}^{\mathrm{Native}}. \tag{6}$$

These guarantees ensure that residual completion retains every source line and complete closure already present in Native. KEEP remains the default when no addition fits. Source preservation and budget safety follow from the construction; the magnitude of any completeness improvement must be established empirically. Appendix I gives the complete procedure.

The development-informed evaluation includes 763 unique defects from SWE-bench Verified, Live Lite, and Lite after cross-benchmark deduplication. Across three localization models, this yields 2,289 model–defect pairs, with each model supplying its own Native base. All defects have repeat-consistent traces and nonempty reference-mapped Python line sets (Appendix N.3). The residual caps

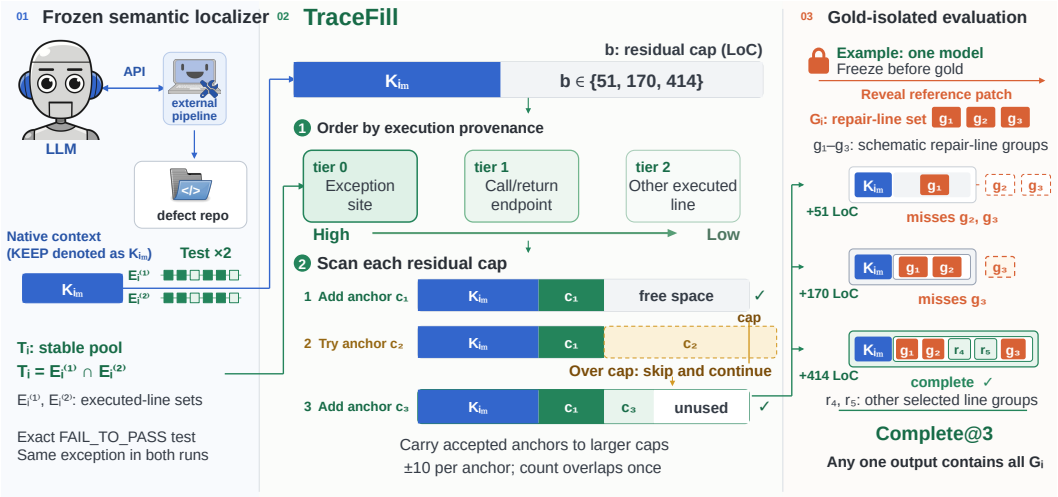


Figure 4: Schematic TRACEFILL construction for one defect–model cell. Native anchors K_{im} are preserved; repeat-consistent executed-line anchors are ranked and added by scan-and-skip under residual cap b. Each anchor renders a clipped ± 10-line window with overlaps counted once. Here g_1–g_3 are schematic groups of reference-mapped repair lines and r_4–r_5 are other selected line groups. Outputs are frozen before gold scoring.

follow median rendered additions from the 3-, 12-, and 34-candidate development configurations. Downstream repair uses a separate 1,004-defect roster in Section 5.3.

With up to 414 additional source lines, TRACEFILL raises Complete(414) from 33.38% for Native to 41.24%, an absolute gain of 7.86 percentage points. The repository-clustered 95% bootstrap confidence interval for this gain is 5.80–11.00 percentage points. Model-specific gains range from 6.55 to 8.78 percentage points, with all three intervals excluding zero (Table 27, Appendix J). Improvements occur in all 27 reported model–dataset–budget combinations (Table 3). At residual caps of 51 and 170 lines, the pooled gains over Native are 2.53 and 4.81 percentage points, respectively. At every evaluated cap, no context complete under Native becomes incomplete after augmentation, as guaranteed by KEEP preservation.

Table 3: Complete(b) (%) by model, dataset, method, and cap. Blue-shaded cells highlight within-model column maxima. Pooled Native/TraceFill use 2,289 pairs; post-hoc Exact-LoC uses 2,001. Table 29 gives cell-specific denominators; Table 27 aligns 1,965 common pairs.

		Verified			Live Lite			Lite		
Model	**Method**	**51**	**170**	**414**	**51**	**170**	**414**	**51**	**170**	**414**
DeepSeek V3.2	Native	18.03	18.03	18.03	11.81	11.81	11.81	23.08	23.08	23.08
	TraceFill	**19.44**	**21.97**	**26.76**	**14.76**	**16.97**	**18.45**	**26.15**	**28.72**	**32.82**
	Same-file	18.41	18.13	19.83	12.08	12.45	13.58	23.71	23.71	23.71
	Anchor-count	18.03	18.03	18.03	11.90	11.90	12.64	23.20	23.20	23.20
	Exact-LoC	17.73	17.73	17.73	11.67	12.06	12.06	21.43	22.08	21.43
Qwen3.5 Plus	Native	60.28	60.28	60.28	34.69	34.69	34.69	68.72	68.72	68.72
	TraceFill	**62.82**	**63.38**	**65.07**	**37.64**	**39.85**	**41.70**	**71.79**	**73.85**	**76.41**
	Same-file	61.19	60.62	61.19	36.47	35.34	36.47	68.72	69.23	69.74
	Anchor-count	60.17	60.17	60.17	34.21	34.21	34.59	68.75	68.75	68.75
	Exact-LoC	59.40	59.73	59.73	34.48	34.48	34.87	71.34	71.34	71.34
GPT-5.3 Codex	Native	27.89	27.89	27.89	32.84	32.84	32.84	35.38	35.38	35.38
	TraceFill	**29.86**	**32.39**	**37.46**	**35.06**	**38.38**	**40.59**	**38.46**	**41.03**	**44.62**
	Same-file	28.25	28.25	28.25	34.08	33.71	35.21	35.90	36.92	35.38
	Anchor-count	27.97	27.97	27.97	32.44	32.44	32.44	35.38	35.38	35.90
	Exact-LoC	26.00	26.33	26.00	33.33	33.33	33.72	34.42	34.42	34.42

TRACEFILL also completes contexts that already contain Native source code. In a post-hoc analysis of each localizer's nonempty Native bases, Complete(414) improves over Native by 5.97–6.62 percentage points across the three models. These strata account for 101 of the 180 additional

complete model–defect pairs (Appendix J). Nonempty Native denotes available source context, without certifying the correctness or completeness of its localization. A separate post-hoc analysis yields observed gains of 9.54 percentage points for single-file defects and 1.29 for cross-file defects.

Matched alternatives test whether other additions achieve comparable containment within the same residual capacity. Three post-hoc restricted controls preserve Native and the cap: Same-file and repository-wide Exact-LoC match rendered size, while Anchor-count matches added-anchor counts. Same-file also uses files receiving TRACEFILL additions. All exclude Native and stable-trace anchors, although rendered windows may overlap executed lines (Appendix I). On the common subset of 1,965 model–defect pairs, TRACEFILL achieves 40.10% COMPLETE@3. It exceeds Same-file by 5.39 percentage points, Anchor-count by 7.53, and Exact-LoC by 7.33 (Table 27). Using a common cohort keeps the compared defect–model pairs fixed. These comparisons favor TRACEFILL's additions over the matched alternatives, but the restricted pools do not isolate provenance ordering or establish superiority over unrestricted matched-budget sampling.

5.2 ABLATIONS AND EXECUTION STABILITY

Native provides the foundation for the observed completion gains. COMPLETE@3 is 41.24% with TRACEFILL, compared with 33.38% for KEEP alone and 20.80% when Native is removed (Table 4a). These comparisons support supplementing the semantic base under the tested policy.

Given this base, priority ordering determines which candidates receive limited capacity. Relative to the full TRACEFILL policy, hash ordering reduces COMPLETE@3 by 5.99 percentage points, while stopping at the first overflow reduces it by 0.79 percentage points. Continuing the scan allows later windows with smaller marginal footprints to fit. These retrospective variants do not establish independent causal contributions or compare optimized alternatives.

Table 4: Ablations and repeat consistency. Panel (a) reports same-model COMPLETE@3 and losses computed before rounding; panel (b) reports run-to-run variation over physical defects. Appendix J defines the execution-status and line-set counts.

Variant	Complete@3 (%)	Loss (pp)
TraceFill	**41.24**	–
KEEP only	33.38	7.86
No KEEP	20.80	20.45
Hash ordering	35.26	5.99
Stop on first overflow	40.45	0.79

(a) Ablation analysis

Execution result	Cases	Different line sets	Lines seen in only one run
Same exception; retained	789	28 (3.55%)	579
Different exceptions; not used	190	79 (41.58%)	27,609
Invalid execution; not used	24	9 (37.50%)	135,431

(b) Repeat-consistency analysis

Across 789 trace-qualified defects, the intersection excludes a total of 579 source-line candidates observed in only one run (Table 4b). On the 763-defect closure-evaluation cohort, either individual trace, their union, and their intersection yield the same aggregate COMPLETE@3 of 41.24%. This conditional comparison shows no additional containment benefit from intersection filtering after two-run qualification. The admission performance and end-to-end cost of a single-trace policy remain unmeasured.

5.3 DOWNSTREAM REPAIR: OVERALL GAINS AND NATIVE AVAILABILITY

Repair success requires a separate evaluation beyond closure containment. The downstream comparison uses shared DeepSeek-V3.2 localization and a 414-line addition cap on 1,004 defects per generator (Jimenez et al., 2024). Across the three repair generators, absolute gains over Native range from 0.60 to 1.69 percentage points (Table 5a). Each arm requests one sample, with pair-symmetric technical failures scored zero. Native supplies no source code for 798 of the 1,004 defects (79.48%), although issue descriptions and repair instructions remain.

On the post-hoc closure-aligned subset of 763 defects per generator, TRACEFILL increases the number of resolved defects from 16 to 22 for DeepSeek, from 41 to 58 for Qwen, and from 52 to 68 for GPT, relative to Native. The 215 trace-gate failures have zero net change. The aligned subset excludes these failures and 26 empty mapped closures from the full roster.

Table 5: Repair outcomes with shared DeepSeek localization. (a) Success on 1,004 defects per generator; post-hoc 95% repository-block bootstrap intervals. (b) Stable Native strata; gain/loss is policy-only/Native-only success.

(a) Overall repair success and relative improvement

Repair generator	Native (%)	Policy (%)	Δ (pp)	Relative gain	95% CI (pp)
DeepSeek-V3.2	1.99	**2.59**	+0.60	+30.0%	[0.23, 1.22]
Qwen3.5-Plus	5.18	**6.87**	+1.69	+32.7%	[0.61, 3.38]
GPT-5.3-Codex	6.77	**8.37**	+1.59	+23.5%	[0.55, 3.47]

(b) Paired outcomes by Native availability

Repair generator	Native state	n	Resolved Native → policy	Gain / Loss	Δ (pp)
DeepSeek-V3.2	Empty	583	$0 \rightarrow 6$	6 / 0	+1.03
	Nonempty	206	$17 \rightarrow 17$	0 / 0	0.00
Qwen3.5-Plus	Empty	583	$6 \rightarrow 19$	17 / 4	+2.23
	Nonempty	206	$36 \rightarrow 40$	12 / 8	+1.94
GPT-5.3-Codex	Empty	583	$14 \rightarrow 30$	23 / 7	+2.74
	Nonempty	206	$39 \rightarrow 39$	7 / 7	0.00

Most of the net increase occurs where Native source context is empty (Table 5b). Within the trace-stable cohort, the empty-Native stratum accounts for net increases of 6, 13, and 16 resolved defects for DeepSeek, Qwen, and GPT, respectively. In the nonempty stratum, DeepSeek resolves 17 defects in each arm. Qwen records 12 gains and 8 regressions, yielding a net increase of 4 resolved defects. GPT's 7 gains are offset by 7 regressions. These individual regressions show that preserving source lines does not guarantee preservation of repair success.

Interface failures contribute to the prevalence of empty Native contexts. Within the trace-stable empty-Native stratum, 567 of the 581 responses rejected at the first localization stage contain valid repository Python paths in formats rejected by the parser. This identifies interface incompatibility without establishing localization correctness. The post-hoc results characterize the recorded shared-localizer configuration, without attributing repair gains causally to closure containment. Corrected-parser and generator-specific-localizer performance remains unmeasured (Appendix J.3).

6 Limitations and Conclusion

Limitations. Mapping remains reference-relative and does not normalize renames. Correcting stale file state changes two gold sets but no Complete labels; restoring filtered pure-addition anchors changes the fixed-panel gain from 7.86 to 7.78 points (Appendix N.3). Nonempty Native indicates availability, not correctness. RQ1/CFRR concern single-file, on-path development defects. Unverified CFRR input equivalence precludes causal ranking attribution; TraceFill does not establish recovery of its 26 Realization cases. Qualifying traces may omit dependencies; actual multi-selector execution remains unrecovered. Post-hoc controls and ablations cannot isolate provenance or repeat-consistency effects, establish discarded lines' irrelevance, or attribute repair gains to containment. Standard SBFL suspiciousness from fail/pass spectra could replace provenance ordering while fixing Native, candidates, ± 10 rendered windows, residual caps, and scan-and-skip; this comparison remains unevaluated. No KEEP is not an optimized execution-only baseline. Development-overlap exclusion is a sensitivity; three models and overlapping benchmarks limit generalization. End-to-end timing and cost remain unmeasured.

Conclusion. Repair-context completeness measures reference-mapped edit-location containment within a source-line budget. RQ1–RQ2 separate candidate availability from complete-context selection; CFRR attributes failures, with Realization the largest first-failure category in the audited development cohort. TraceFill preserves Native and improves containment through execution-derived residual completion across the evaluated model panel.

Reproducibility Statement. Packaged hashes and ledgers support deterministic reaggregation of the reported summaries. Appendix K documents provenance, immutable run identifiers, and the offline procedure.

REFERENCES

Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. In *Proceedings of PLDI*, pp. 246–256, 1990. doi: 10.1145/93542.93576.

Mohammad Nour Al Awad and Sergey Ivanov. Loc2Repair: A framework for evaluating the impact of file-level issue localization in repo-level LLM repair. *arXiv preprint arXiv:2606.30963*, 2026. URL https://arxiv.org/abs/2606.30963.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Yang Wang. SWE-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a1e6783e4d739196cad3336f12d402bf-Abstract-Conference.html.

Fatmah Yousef Assiri and James M. Bieman. Fault localization for automated program repair: Effectiveness, performance, repair correctness. *Software Quality Journal*, 25(1):171–199, 2017. doi: 10.1007/s11219-016-9312-z.

Prantik Chatterjee, Jose Campos, Rui Abreu, and Subhajit Roy. Augmenting automated spectrum based fault localization for multiple faults. In *Proceedings of IJCAI*, pp. 3140–3148, 2023. doi: 10.24963/ijcai.2023/350.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In *Proceedings of ACL*, pp. 8697–8727, 2025. doi: 10.18653/v1/2025.acl-long.426.

Mahdi Farzandway and Fatemeh Ghassemi. SieveFL: Hierarchical runtime-aware pruning for scalable LLM-based fault localization. *arXiv preprint arXiv:2605.13491*, 2026. URL https://arxiv.org/abs/2605.13491.

Mirazul Haque, Petr Babkin, Farima Farmahinifarahani, and Manuela Veloso. Towards effectively leveraging execution traces for program repair with code LLMs. In *Proceedings of the 4th Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pp. 160–179, 2025. doi: 10.18653/v1/2025.knowledgenlp-1.17.

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*, 2024. doi: 10.48550/arXiv.2310.06770.

James A. Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of ICSE*, pp. 467–477, 2002. doi: 10.1145/581339.581397.

Kui Liu, Anil Koyuncu, Tegawende F. Bissyande, Dongsun Kim, Jacques Klein, and Yves Le Traon. You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. In *Proceedings of ICST*, pp. 102–113, 2019. doi: 10.1109/ICST.2019.00020.

Yiling Lou, Ali Ghanbari, Xia Li, Lingming Zhang, Haotian Zhang, Dan Hao, and Lu Zhang. Can automated program repair refine fault localization? a unified debugging approach. In *Proceedings of ISSTA*, pp. 75–87, 2020. doi: 10.1145/3395363.3397351.

Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI software engineering with repository-level code graph. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/4a4a3c197deac042461c677219efd36c-Abstract-Conference.html.

Nikhil Parasaram, Huijie Yan, Boyu Yang, Zineb Flahy, Abriele Qudsi, Damian Ziaber, Earl T. Barr, and Sergey Mechtaev. The fact selection problem in LLM-based program repair. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*, pp. 2574–2586, 2025. doi: 10.1109/ICSE55347.2025.00162.

Spencer Pearson, Jose Campos, Rene Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In *Proceedings of ICSE*, pp. 609–620, 2017. doi: 10.1109/ICSE.2017.62.

Revanth Gangi Reddy, Tarun Suresh, JaeHyeok Doo, Ye Liu, Xuan-Phi Nguyen, Yingbo Zhou, Semih Yavuz, Caiming Xiong, Heng Ji, and Shafiq Joty. SWERank: Software issue localization with code ranking. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7f6901ebab786e43b21530328fc989ca-Abstract-Conference.html.

Bat-Chen Rothenberg and Orna Grumberg. Must fault localization for program repair. In *Computer Aided Verification*, pp. 658–680, 2020. doi: 10.1007/978-3-030-53291-8_33.

Melika Sepidband, Hung Viet Pham, and Hadi Hemmati. On the role of fault localization context for LLM-based program repair. *arXiv preprint arXiv:2604.05481*, 2026a. URL https://arxiv.org/abs/2604.05481.

Melika Sepidband, Hamed Taherkhani, Hung Viet Pham, and Hadi Hemmati. RGFL: Reasoning guided fault localization for automated program repair using large language models. *arXiv preprint arXiv:2601.18044*, 2026b. URL https://arxiv.org/abs/2601.18044.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. CoRNStack: High-quality contrastive data for better code retrieval and reranking. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/285b0e63256699c04dbf2c8d4163fba4-Abstract-Conference.html.

Boshi Wang, Weijian Xu, Yunsheng Li, Xuemei Gao, Yujia Xie, Huan Sun, and Dongdong Chen. Improving code localization with repository memory. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b4c06f095368497f3ac19422efef8133-Abstract-Conference.html.

W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016. doi: 10.1109/TSE.2016.2521368.

Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. CrashLocator: Locating crashing faults based on crash stacks. In *Proceedings of ISSTA*, pp. 204–214, 2014. doi: 10.1145/2610384.2610386.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025. doi: 10.1145/3715754.

Qi Xin, Haojun Wu, Jinran Tang, Xinyu Liu, Steven P. Reiss, and Jifeng Xuan. Detecting, creating, repairing, and understanding indivisible multi-hunk bugs. *Proceedings of the ACM on Software Engineering*, 1(FSE):2747–2770, 2024. doi: 10.1145/3660828.

Hui Xu, Zhaodan Wang, and Weiqin Zou. A more accurate bug localization technique for bugs with multiple buggy code files. *Information and Software Technology*, 181:107675, 2025. doi: 10.1016/j.infsof.2025.107675.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. doi: 10.48550/arXiv.2405.15793.

He Ye and Martin Monperrus. ITER: Iterative neural repair for multi-location patches. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–13, 2024. doi: 10.1145/3597503.3623337.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of ISSTA*, pp. 1592–1604, 2024. doi: 10.1145/3650212.3680384.

AI USE STATEMENT

OpenAI Codex assisted with literature retrieval and discovery and with proving mathematical claims. The authors verified the retrieved sources and mathematical arguments and take responsibility for the final text and claims.

A Extended Related Work

Repository-level localization and context selection. Repository-level localization and repair systems use hierarchical retrieval, graph navigation, memory, code ranking, and interactive search to select code before generation (Xia et al., 2025; Chen et al., 2025; Yang et al., 2024; Zhang et al., 2024; Ouyang et al., 2025; Suresh et al., 2025; Antoniades et al., 2025; Reddy et al., 2026; Wang et al., 2026). RGFL examines reasoning-guided localization and a counterfactual localization upper bound (Sepidband et al., 2026b). Set completeness is already explicit in SWERank and RepoMem; RepoMem's file-level Accuracy@k requires every ground-truth file, and both study downstream resolution (Reddy et al., 2026; Wang et al., 2026). Our distinction is the scored object and budget: full pre-fix line containment in one rendered context under a declared source-line limit, coupled with failure-stage attribution and residual completion. Table 7 details the mechanisms and contracts rather than a performance ranking.

Execution evidence for fault localization. Spectrum-based localization, dynamic slicing, and crash localization use runtime evidence to rank suspicious statements or propagate dependencies (Agrawal & Horgan, 1990; Jones et al., 2002; Wu et al., 2014; Wong et al., 2016). AutoCodeRover augments AST-based search with test spectra when available; SieveFL uses coverage to prune methods before LLM screening and reranking (Zhang et al., 2024; Farzandway & Ghassemi, 2026). RQ2 tests specified uses of execution for direct context assembly, distinguishing exact-test from whole-module-proxy protocols and selection from response and certification. Tables 6 and 7 position these mechanisms alongside learned ranking and residual completion.

Table 6: Stage-wise positioning relative to upstream localization systems. The table compares interfaces and endpoints; cross-system integration is not evaluated.

Work	Stage	Input → output	Primary endpoint	Relation to this work
RGFL	Upstream localization	Issue and repository → ranked code locations	Localization and repair performance	Can supply the semantic candidates to a completion policy.
SieveFL	Upstream pruning and ranking	Failing-test coverage and methods → ranked suspicious methods	Fault-localization performance	Can prune or supply the candidate pool before assembly.
This work	Context assembly	Upstream context and fixed candidates → bounded repair contexts	Complete(b), same-model COMPLETE@3, and descriptive repair outcomes	Tests whether available regions are selected together in one call.

Table 7: Qualitative contract-level comparison with selected related work, not a matched-budget performance comparison or integration study. "Not evaluated" describes a study endpoint, not a method's potential capability.

Work	Output and reported endpoint	Joint-context and budget contract	Attribution and execution role
Agentless	Hierarchical edit-location sets and concatenated windows; Contains-GT, average LoC, and repair outcomes.	A sampled set may place multiple windows in one repair prompt, while four sampled sets are repaired separately. Fixed ± 10 windows are used, but no hard total rendered-line endpoint is reported.	Component ablations do not form a mutually exclusive first-failure audit. Execution is used downstream for patch validation and selection rather than trace-selected context completion.
MANIPLE	A selected subset of static, dynamic, and external facts plus a provided buggy function; predicted success, Pass@1, and repair correctness.	Function-granular perfect localization is assumed; multi-region source containment and a fixed rendered-source-line limit are not evaluated.	No context-construction failure attribution. Execution-derived facts are prompt inputs, rather than coverage-selected source regions.
HitMore	A ranked file list; `multiCompleteness@All` asks whether a top-K list contains every modified or deleted buggy file.	Establishes set completeness at file granularity, but does not construct one repair call or impose a rendered-source-line limit.	Staged file retrieval is evaluated, not CFRR-style first-failure attribution. The method uses static IR, dependence, and history signals.

Continued on next page

Table 7 (continued)

Work	Output and reported endpoint	Joint-context and budget contract	Attribution and execution role
ITER	A pool of plausible patches accumulated through repeated localization, generation, and validation; correct and plausible repairs, including multi-location bugs.	Joint edits are assembled across iterations rather than evaluated as one frozen, bounded repair context. No rendered-source-line containment endpoint is reported.	Test execution validates partial patches and can trigger re-localization; no mutually exclusive context-failure decomposition is defined.
Execution-trace prompting	A buggy-function prompt augmented with failing-test feedback and execution traces; Correct Fix and Correct Program Accuracy.	The buggy function is supplied, so repository-region selection and closure containment under a source-line limit are not evaluated.	Trace representations are inserted directly into generation prompts; no localization-stage attribution is defined.
AutoCodeRover (Zhang et al., 2024)	Iteratively retrieved code and generated patches; issue resolution.	Retrieves context for patch generation rather than filling a frozen Native base. We do not compare its outputs under common rendered-source-line caps.	AST-based class and method search; test spectra can guide retrieval when tests are available.
SieveFL (Farzandway & Ghassemi, 2026)	Ranked suspicious methods; Top-k localization accuracy and MRR.	Prunes method candidates before ranking. We do not evaluate this policy under TraceFill's KEEP constraint and residual-source-line caps.	Failing-test coverage prunes unexecuted methods before individual LLM screening and comparative reranking.
SWERank (Reddy et al., 2026)	Ranked code locations; all-relevant-location Acc@k and downstream repair.	A top-k cutoff counts code units rather than unique rendered lines. Its outputs are not evaluated under our residual-cap contract.	Learned issue-to-code retrieval and reranking trained on SWELoc; a ranking alternative beyond generic addition controls.
This work	One frozen, rendered repair context; Complete(b) and same-model COMPLETE@3.	Tests $G_i \subseteq W_i(A)$ under a declared total source-line limit for a single repair call.	CFRR attributes exposure, reachability, capacity, and realization failures; TraceFill uses execution only for gold-blind residual completion.

Localization–repair interface and context utility. Repair-aware studies show that localization choices and evaluation protocols affect reported localization and repair outcomes (Assiri & Bieman, 2017; Liu et al., 2019; Lou et al., 2020; Pearson et al., 2017). Loc2Repair isolates file localization as an upstream repair variable, while recent context analysis varies file-, element-, and line-level inputs (Al Awad & Ivanov, 2026; Sepidband et al., 2026a). At the localization–repair interface, TRACEFILL's post-hoc controls examine file scope, anchor structure, and added source lines separately.

Multi-region and repair-aware objectives. Multi-fault and multi-hunk studies show that localization and repair may involve multiple noncontiguous locations (Chatterjee et al., 2023; Xin et al., 2024). Our endpoint asks whether one budgeted context jointly contains all regions mapped from one observed patch. Must fault localization seeks a set intersecting every repair admitted by a repair scheme (Rothenberg & Grumberg, 2020). Reference-patch closure measures full containment of one observed patch; alternative repairs remain separate outcomes. CFRR measures exposure, reachability, capacity fit, and gold-blind realization separately, so oracle headroom is reported as a diagnostic.

B EVIDENCE SCOPE AND REPRODUCIBILITY

This appendix distinguishes the evidentiary role and reproducibility status of each analysis. RQ2 separates direct context selection, candidate availability, behavioral response, and directional certification; CFRR and residual-completion analyses retain their own cohorts and contracts. Response and certification rates describe mechanism probes, while gold-aware ceilings remain diagnostic upper bounds.

Artifact contents. The anonymous supplementary package supports deterministic reaggregation of the RQ2 primary and RQ4 summaries from processed ledgers. It includes test selectors, code and input hashes, seeds, environment metadata, fixed action files, the sample funnel, and gold-firewall metadata. RQ2 supporting experiments are documented through aggregate summaries and provenance records. The reaggregation procedure operates on these processed records.

C EVALUATION CONTRACT AND GOLD-BLINDNESS

This appendix fixes the shared gold, rendering, and freeze contract used by the later analyses. The canonical gold object is the pre-fix line mapping obtained from the reference patch after lexical

cleaning (`clean_gold` in the implementation). It represents the observed reference patch, while semantically valid alternatives form separate repair outcomes. Every final anchor receives exactly one ± 10 expansion, clipped to the pre-fix file; overlapping lines are unioned. The mapper handles addition-only hunks, renames, and deletions according to the artifact's canonical rules. RQ1 and CFRR score the rendered union of Native's frozen localization anchors. RQ4 closure evaluation fixes one Native base per model and defect, then scores each of three budget-specific completions without merging them. Containment and behavioral repair success remain separate endpoints.

Operational scope of the mapping. For the closure-containment matrix, Appendix N.3 specifies the original Python edit-location rule, its audited file-boundary correction, and a separate anchor-restoration sensitivity. The correction leaves the reported Complete labels and denominators unchanged. Lexical cleaning and unsupported path forms still limit the reference-relative endpoint; it does not guarantee a complete representation of every patch.

Candidate generation is gold-blind. Before scoring, the generation pipeline freezes instance IDs, eligibility rules, semantic anchors, dynamic candidate traces, action budgets, random seeds, matched-control construction, and rendered footprints; the artifact records the available bindings and hashes for these frozen objects. Completeness is then scored against the reference closure. In the repeated exact test, the prespecified primary analysis filters to baseline-incomplete rows after this freeze. Gold-aware minimum unions serve as labeled CFRR diagnostics. Table 8 applies this common contract to every research question.

Table 8: Unified evidence contract. Gold scores frozen outputs after generation, ranking, sizing, and selection. Supporting sample counts refer separately to Experiments 2–6; Appendix G.3 distinguishes reachability and matched samples.

RQ	Cohort	n	Trace / evidence	Gold role	Endpoint and status
RQ1	Curated development	117	Fixed Native context	Score only	Native context completeness; descriptive boundary association.
RQ2 exact	Frozen roster	126 eligible; 101 primary	Repeated exact F2P	Score + prespecified baseline filter	Primary matched comparison.
RQ2 earlier	Dev. intersection	17	Exact F2P	Post-freeze score	Mechanism precursor.
RQ2 support	Supporting ledgers	61; 57/61; 9; 18 valid (17 paired); 17	Proxy, structural, label, and intervention evidence	Post-freeze score	Supporting mechanism evidence.
CFRR	Dev. universe	117	Whole-module proxy	Gold-aware ceiling	Reachability diagnostic over a frozen action family.
RQ4 closure	3 benchmarks $\times$ 3 models	763 defects; 2,289 model–defect pairs	Repeated exact F2P	Score only	Any-budget containment within each model across residual added-LoC caps $b \in \{51, 170, 414\}$; repository-clustered uncertainty.

D Supplementary Roadmap and Claim–Evidence Matrix

This roadmap assigns a distinct role to each remaining appendix. Appendix E defines the repair-context completeness objects, denominators, and pairing rules; Appendices F and G report the structural analysis and six direct-execution experiments; and Appendices H and I specify CFRR and the deterministic TraceFill policy. Appendix J reports attrition, budgets, controls, model–benchmark cells, and downstream outcomes. Appendices K and L provide run-level reproduction coordinates and the source–data index, while Appendices M and N state scope limits and recorded input-construction details. Table 9 lists the denominator and evidentiary role of every central result.

Table 10 defines the observables and interpretation of each endpoint.

Table 9: Claim–evidence matrix. "Primary" denotes the main reported endpoint within an RQ; RQ4 aggregates containment scores across budgets after outputs are frozen. "Diagnostic" denotes mechanism or headroom measured over frozen actions.

Claim	Measured object	Unit / denominator	Comparator or control	Evidence status
Problem formulation / RQ1	One context contains the full closure	117 development defects	Exposed/omitted and structural groups	Descriptive repair-context boundary association.
Execution boundary / RQ2	Frozen execution-derived action completes the closure	101 baseline-incomplete defects	1,000 within-instance matched controls	Primary controlled test of direct crash-local selection.
RQ2 mechanisms	Breadth, structure, labels, response, direction	61; 57/61; 9; 18 valid (17 paired); 17	Matched permutations, shams, opposites	Experiment-specific mechanism endpoints.
RQ3 CFRR	Exposure, reachability, fit, realization	117 development defects	Nested contract stages	Stage-wise diagnostic and upper-bound audit.
RQ4 instantiation	Any-budget containment within each model at residual caps $b \in \{51, 170, 414\}$	763 defects across 3 models; 2,289 descriptive pairs	Native and aligned structural controls	Post-freeze three-budget completion reaggregation.

Table 10: Endpoint separation. Each quantity answers a different question and is reported on its own denominator.

Endpoint	Observable	Interpretation
File exposure	All repair files are upstream-visible	Necessary availability condition preceding line selection.
Candidate reachability	Some fixed action contains the closure	Gold-aware diagnostic over a gold-blind action family.
Capacity fit	A reachable action fits the declared budget	Separates candidate supply from rendered capacity.
COMPLETE@3	One of three same-model, budget-specific contexts contains all closure lines	Post-freeze localization endpoint over 2,289 descriptive panel rows; inference resamples repositories.
Resolved / Pass@1 / behavioral correctness	When a patch is available, the sole patch passes the official harness exactly when the instance is resolved; technical failures score zero	One-request-per-arm downstream endpoint; the three names share one binary observation.

E FORMAL MEASUREMENT SPECIFICATION

E.1 CANONICAL CLOSURE AND RENDERED FOOTPRINT

Let P_i be the observed reference patch for defect i. Under the declared path, language, and lexical-filtering rules, G_i is the union of surviving pre-fix Python edit-location anchors; it need not retain an anchor from every hunk of P_i. Empty canonical Python closures are excluded uniformly after gold construction and before any model- or arm-specific comparison. For an anchor set A, let $E_i(a)$ denote the clipped one-time ± 10 expansion of anchor a in the pre-fix file. The rendered unique-line footprint is

$$W_i(A) = \bigcup_{a \in A} E_i(a), \qquad \tau_i(A) = |W_i(A)|. \tag{7}$$

Each final anchor is expanded once, and overlapping windows are unioned before cost is measured. The capacity unit is rendered source lines.

For residual source-line caps $\mathcal{B} = \{51, 170, 414\}$, let A_{imb} be the anchor set for defect i, localization model m, and cap b, suppressing the arm index. Let $\mathcal{P}$ be the eligible defect–model pairs for the specified analysis, fixed across all three caps. We score each rendered context separately:

$$\begin{aligned} C_{imb} &= \mathbf{1}[G_i \subseteq W_i(A_{imb})], \quad \text{Complete}_{\mathcal{P}}(b) = \frac{1}{|\mathcal{P}|} \sum_{(i,m)\in\mathcal{P}} C_{imb}, \\ C_{im}^{@3} &= \max_{b \in \mathcal{B}} C_{imb}, \qquad\qquad \text{Complete@3}_{\mathcal{P}} = \frac{1}{|\mathcal{P}|} \sum_{(i,m)\in\mathcal{P}} C_{im}^{@3}. \end{aligned} \tag{8}$$

Pairs receive equal weight. The balanced Native–TraceFill panel has $\mathcal{P} = \mathcal{I} \times \mathcal{M}$ and $|\mathcal{P}| = 3|\mathcal{I}| = 2,289$. Each comparator-specific set requires both arms to be available at all three caps and to satisfy the declared matching rules; the all-control comparison uses their common intersection. Both arms use the same pair set, which also remains fixed between single-cap and any-budget summaries.

The maximum compares three separately rendered contexts within one localization model; it does not merge them. Native and TraceFill have nested contexts, making COMPLETE@3 equal to Complete(414). For non-nested controls, COMPLETE@3 is the upper envelope of three frozen budget-specific outcomes, not an expectation over random seeds or an implemented gold-blind choice among contexts. On a fixed comparison cohort, this aggregation cannot lower a control's score relative to its own Complete(414). Separate construction does not imply statistical independence. Table 29 reports single-cap cells; Table 27 additionally compares Complete(414) and COMPLETE@3 on the same 1,965 control-matched pairs. Downstream evaluation uses one repair request per arm.

E.2 COMPLEMENTARY REPAIR-REGION PREDICTION

A completion policy f_θ takes the frozen Native anchor set S_i, stable execution evidence, optional dependency evidence, and a source-line budget B_i, suppressing model and budget indices. It selects complementary anchors Δ_i and forms

$$\widehat{S}_i = S_i \cup \Delta_i, \qquad \tau_i(\widehat{S}_i) \le B_i. \tag{9}$$

The union construction implies KEEP preservation. Rendering $\widehat{S}_i$ once gives the augmented context $W_i(\widehat{S}_i)$ within the source-line limit. The reference patch, canonical closure, and gold-derived outcome labels are excluded from candidate generation, ranking, and packing; scoring reads them only after outputs are frozen. Legitimate source and execution inputs may contain lines later identified by the reference mapping. The post-freeze objective is $\mathbf{1}[G_i \subseteq W_i(\widehat{S}_i)]$. TraceFill instantiates this completion problem with a deterministic selection rule.

E.3 PAIRING, WINS, LOSSES, AND FIXED-PANEL ESTIMANDS

For containment arms a and a' on a fixed eligible pair set $\mathcal{P}$, define $Y^a_{im}, Y^{a'}_{im} \in \{0,1\}$ after applying COMPLETE@3 within localization model m, and let $n = |\mathcal{P}|$. The paired risk difference and discordant counts are

$$\begin{aligned}
\widehat{\Delta}_{a,a'} &= \frac{1}{n} \sum_{(i,m)\in\mathcal{P}} (Y^a_{im} - Y^{a'}_{im}), \\
N_{\text{win}} &= \sum_{(i,m)\in\mathcal{P}} \mathbf{1}[Y^a_{im} = 1, Y^{a'}_{im} = 0], \\
N_{\text{loss}} &= \sum_{(i,m)\in\mathcal{P}} \mathbf{1}[Y^a_{im} = 0, Y^{a'}_{im} = 1], \qquad N_{\text{tie}} = n - N_{\text{win}} - N_{\text{loss}}.
\end{aligned} \tag{10}$$

Tables abbreviate these counts as W/L/T. Closure pooling first applies the fixed Verified–Lite reuse map so that cross-benchmark aliases contribute once, giving 763 defects and 2,289 descriptive model–defect panel rows. Repository-clustered resampling draws whole repositories, carrying all defects and model rows in each sampled repository together; model-stratified results provide separate estimates for each model. Benchmark–model cells retain logical benchmark membership.

For downstream repair, g indexes the repair generator and Y^a_{ig} is its binary official-harness outcome. The same paired formulas apply over eligible (i, g) pairs, without a budget maximum. Pairing is within defect and generator; it does not denote a generator-aligned end-to-end localization–repair pipeline. Repair summaries retain the full official roster and one request per arm, with technical failures scored zero. Table 11 records the pairing and scoring checks.

Table 11: Scoring invariants enforced before any reported comparison.

Invariant	Operational check	Protected property
Single expansion	Final anchors receive exactly one clipped ± 10 expansion	Exactly-once rendered coverage.
Unique-line cost	Overlaps are unioned before τ_i is measured	Single-counted rendered source cost.
Gold firewall	Candidate pools, rankings, budgets, seeds, and contexts are frozen and hashed before scoring	Gold-blind deployable action selection.
Monotone KEEP	$S_i \subseteq \widehat{S}_i$ for every TraceFill output	Retention of the semantic base.
Pair identity	Dataset, instance, model, and arm configuration agree	Valid pairing and consistent denominators.
Exact-key reconciliation	Every expected cell is present, unique, well formed, and matched	Complete aggregation over the declared key set.
Explicit attrition	Owner exclusion, trace gate, empty closure, and structural mismatch have distinct codes	Separate reporting of eligibility and model outcomes.

F RQ1: Development-Cohort Details

The RQ1 cohort contains 117 canonical single-file, on-path defects: 56 in cohort 1 and 61 in cohort 2. Native is context-complete for 77 defects (Table 12). Table 1 reports all mechanical boundary associations that vary within the cohort. Among the 117 closures, 96 are multi-line, 43 are multi-hunk, 33 are multi-function, and 54 are multi-region; all are single-file.

Table 12: Canonical RQ1/CFRR development ledger. Cohort 1 contains 56 defects and cohort 2 contains 61; all 117 enter the Native completion audit.

Cohort	Defects	Native complete
Cohort 1	56	35
Cohort 2	61	42
Total	117	77

Table 13 and Figure 5 show the repository-specific denominators behind the pooled cohort.

Table 13: RQ1 development-cohort breakdown from the fixed Native localization outputs. Counts are descriptive; the cohort has been repeatedly used for development.

Repository	Evaluable	Native complete
astropy	11	9
django	45	31
matplotlib	17	8
seaborn	3	2
flask	2	2
requests	4	1
xarray	3	2
pylint	4	3
scikit-learn	5	5
sympy	23	14
Total	117	77

With repository sample sizes ranging from two to 45, these rates provide a descriptive comparison of Native context completeness across projects.

The structural labels in Table 14 overlap, whereas Table 15 assigns each failure to its first auditable boundary. The full-cohort comparisons associate line and region composition with Native incompleteness. Separately, the failure accounting attributes 13 cases to upstream repair-file omission and 27 to Native incompleteness after all repair files are exposed.

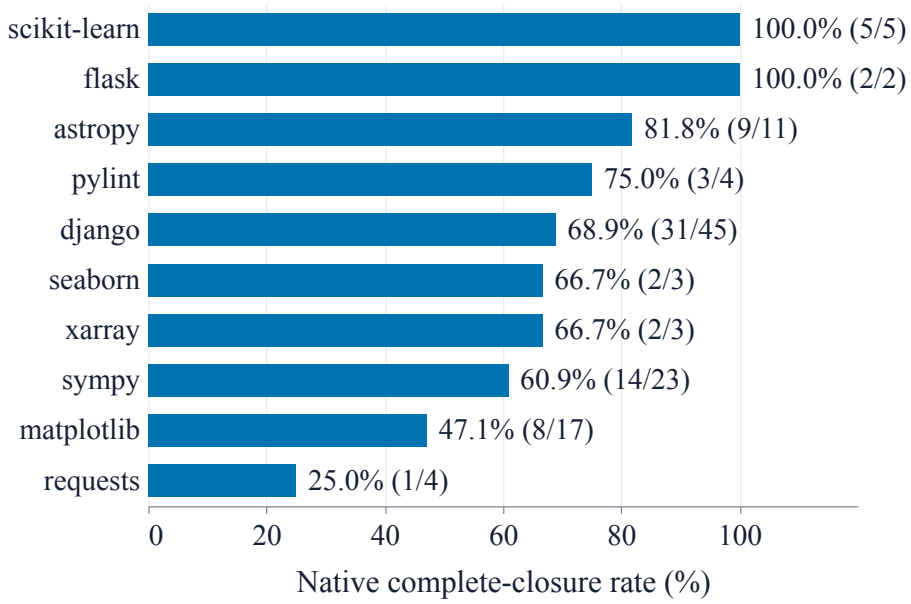


Figure 5: Native context completeness by repository in the 117-defect development cohort. Labels give complete/evaluable counts.

Table 14: Mechanical closure taxonomy used in RQ1. Labels overlap; each contrast is computed against the complementary group on the same 117 defects. The Multi-file and Off crash path zero entries reflect empty categories fixed by cohort design, not estimated zero completion rates.

Boundary	Frozen rule	Positive	Native complete	Measurement role
Repair file exposed	Every file containing a canonical closure line appears in the upstream found-file set	104	77	Tests the upstream availability boundary.
Multi-line	$\|G_i\| > 1$ after canonical mapping and deduplication	96	62	Separates raw closure size from fragmentation.
Multi-hunk	Reference patch contains more than one source hunk	43	17	Captures patch-level separation.
Multi-function	Canonical closure intersects more than one enclosing function	33	11	Captures semantic-unit separation.
Multi-region	Canonical closure has more than one maximal run of consecutive mapped lines, counted per file before ± 10 rendering	54	28	Captures the joint-set composition burden.
Multi-file	Canonical closure spans more than one source file	0	0	Fixed by the single-file RQ1 cohort definition.
Off crash path	Canonical closure is outside the curated crash-path scope	0	0	Fixed by the on-path RQ1 cohort definition.

Table 15: Failure accounting for the Native RQ1 cohort. Counts partition the 40 Native-incomplete outputs only at the first auditable boundary.

First auditable state	Defects	Share of 40 failures
Required repair file omitted upstream	13	32.5%
All repair files exposed but context incomplete	27	67.5%
All Native-incomplete defects	40	100.0%

G RQ2: Evidence for the Direct-Execution Boundary

The six RQ2 tests probe direct selection and its proposed mechanisms through different cohorts, estimands, and controls. Figure 6 summarizes two selection comparisons, and Table 16 maps each experiment to its measured endpoint. Experiment 5 measures behavioral response, and Experiment 6 tests directional certification; each experiment reports its own endpoint and denominator.

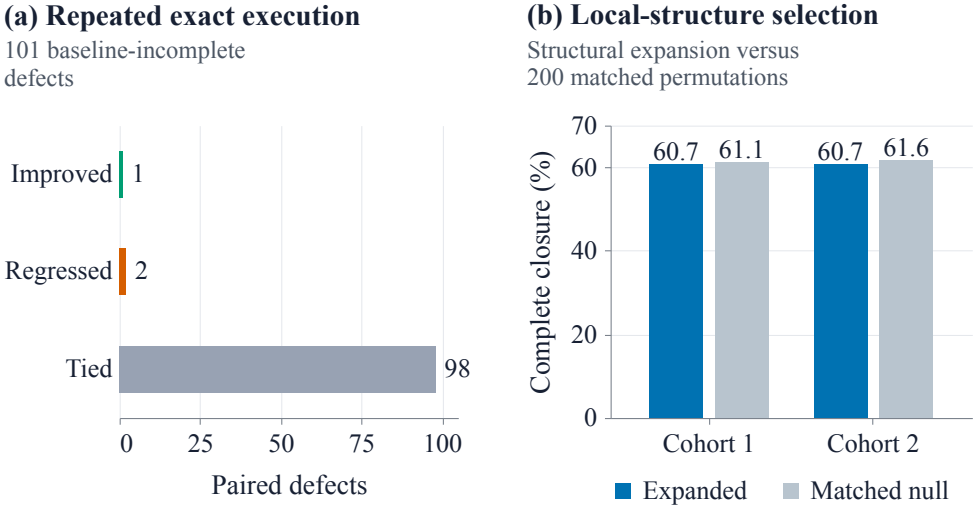


Figure 6: Quantitative RQ2 comparisons: paired outcomes for the exact crash-local action, and budgeted structural expansion compared with matched permutations.

Table 16: Interventions and controls for six alternative explanations of why direct use of execution coverage might select the entire repair closure.

Experiment	Rescue hypothesis	Frozen intervention	Control	Observed result
1	Crash neighborhood is sufficient	Repeat-stable exact-F2P crash action plus KEEP	1,000 matched actions per instance	One improvement, two regressions, and 98 ties.
2	Crash-local pool is too narrow	Full found-file execution pool	Gold-aware ceiling versus frozen selector	Large ceiling–realization gap.
3	Dynamic relations identify companions	Traceback, def–use, and caller tiers	200 permutations matched on tier, file, and count under a shared source-line cap	Structural expansion is 0.40 and 0.90 points below matched selection.
4	Passing coverage is noise	Failing-minus-passing lines	Paired failing- and passing-test support sets	Six of nine closures contain no failing-only gold line.
5	Semantic sites respond more often	Reversible abstract syntax tree (AST) perturbation	File/type/band/scale-matched probes and shams	No detected response advantage: $\Delta Q = -0.029$ ($p = 0.778$).
6	Direction certifies local repair	Predicted predicate force	Opposite force, repeated target execution, sham, restore	No full directional certificate among 32 constraints.

G.1 Experiment 1: Repeated Exact Execution and Crash Proximity

The repeated exact-execution experiment begins from a fixed 219-defect roster. Exact-selector resolution, expected-failure and non-interference checks, and repeated-trace stability yield 126 action-eligible rows across ten repositories. The prespecified baseline-incomplete subset contains 101. Before gold scoring, the experiment froze the KEEP-plus-crash action and its 1,000 within-instance controls. The anonymous package releases the 126 action and score rows and binds each control set by its count and ordered-record digest. Controls preserve Native and the exact per-file incremental-anchor counts; their expanded incremental source-line size must lie within $\pm 20\%$ of the target. For each defect, we subtract the mean containment of its 1,000 controls from the crash-action outcome. The mean difference across defects is +0.008 percentage points, with a repository-clustered 95% confidence interval of $[-0.421, 1.769]$, below the prespecified +5-point practical threshold. One

difference is positive, two are negative, and 98 are zero. A post-hoc supply check finds that 79 actions add no source lines beyond KEEP; Table 17 separates these from the 22 nonempty additions. The primary interval estimates the overall policy on all 101 defects. A post-hoc analysis of the 22 active additions across seven repositories yields $+0.036$ points (repository-clustered 95% CI $[-2.012, 7.718]$). It uses 10,000 whole-repository draws, seed 2026080702, and the original order-statistic percentile rule. Only two repositories have nonzero net differences; an exact two-sided repository sign-flip gives $p = 1.000$ (four of four sign patterns). This tests cluster-sign symmetry, not the unavailable joint distribution of the 1,000 controls. Existing individual randomized-control tails are all at least 0.717; the original 101-case and post-hoc 22-case Holm-adjusted minima are both 1. The conditional interval includes effects above $+5$ points, leaving active selection efficacy unresolved. Figure 8, view (1), shows a crash seed between two required regions.

Two earlier audits provide supporting observations. On the 17-defect development intersection, exact execution makes 16 closures feasible, while the locked crash action completes two. In a further nine-defect crash-selection audit, two closures are completed, but no case combines complete containment with a per-instance matched-null $p \leq 0.05$ ($D = 0$); the aggregate permutation p-value is 0.5821. These audits retain their own denominators and precede the repeated 101-defect primary comparison.

Table 17: Action-supply decomposition for the 101-defect primary set. "Increment" is code added beyond the preserved KEEP context.

Crash-action supply	n	**Improved**	**Regressed**	**Tied**
Zero incremental rendered code	79	0	0	79
Nonzero incremental rendered code	22	1	2	19
All primary defects	101	1	2	98

The anonymous package also documents Experiment 1 trace qualification. Supporting results for experiments 2–6 are organized as cohort-specific aggregate summaries with provenance records.

Primary-interval calculation. The 101-defect RQ2 primary analysis uses 10,000 whole-repository bootstrap draws over ten repositories with seed 2026080702. Each defect contributes its crash-action outcome minus the mean outcome of its 1,000 frozen controls. Each draw pools the defects from ten repositories sampled with replacement. The interval uses zero-based order-statistic indices 250 and 9,750 of the sorted draws, without RQ4's linear interpolation. Zero-increment actions remain in the primary estimand, and the controls are not regenerated within bootstrap draws. The artifact retains these per-defect control means and the analysis configuration; its offline replay reads the stored RQ2 interval rather than recomputing the underlying control scores.

G.2 Experiment 2: Candidate Breadth

Experiment 2 tests whether broader candidate supply improves selection. On development cohort 2 (61 defects), the fixed full found-file candidate pool combines Native and vote anchors, related regions, and whole-module proxy coverage within found files. Its gold-aware closure ceiling is 86.9%. The selector scores candidates by text relevance, structural proximity, definition–use distance, and execution coverage, with weights 1.0, 0.6, 1.2, and 0.8. Text relevance uses issue/failure term overlap with file-local inverse frequencies. Structural proximity is 1 for a seed's function, 0.4 for its callees, and 0 otherwise; the definition–use term decays by 0.6 per hop. The issue contributes its first 5,000 characters; failure text is the 1,600-character traceback prefix, or the last 1,200 problem-statement characters when no traceback is present. Files are ordered by their highest line score, with cached file priority used first in cohort 1. Within each file, candidates are ordered by descending line and cached vote scores, then Native membership and increasing line number. Missing line scores are zero. Selection starts empty, so Native anchors compete with additions. This tests the fixed heuristic under candidate competition, not a policy that preserves Native; displacement is a possible explanation, not an independently isolated causal effect. Each cap receives a fresh scan: accept a clipped ± 10 window if the union of selected source lines fits, otherwise skip it and continue. At the common cap equal to Native's mean rendered size, this policy achieves 42.6% completeness from the expanded pool and 65.6% from Native anchors alone. Figure 7 and Table 18 give the sweep; Figure 8, view (2), illustrates incomplete selection from an available closure.

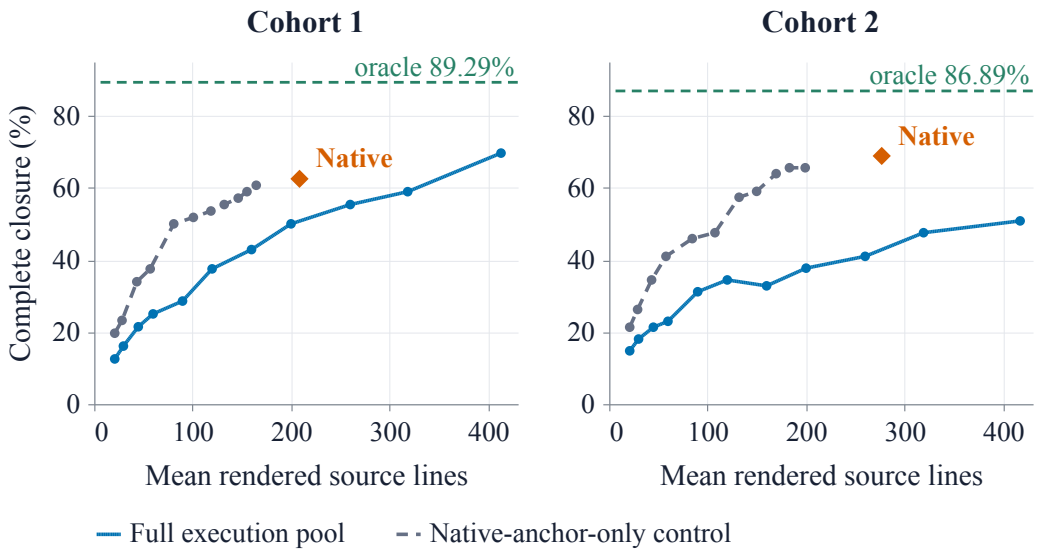


Figure 7: Raw gold-blind budget curves for the 56 cost-valid cohort 1 defects and 61 cohort 2 defects in the frozen selector ledger. The solid curve uses the full execution-augmented pool; the dashed curve applies the same policy to Native anchors. Diamonds mark Native operating points, horizontal rules mark gold-aware full-pool ceilings, and segments join observed points.

Table 18: Full-pool context-complete rate across frozen source-line limits. LoC rows report mean rendered unique source lines.

Cohort	21	30	45	60	90	120	160	200	260	320	420
Cohort 1 (%)	12.50	16.07	21.43	25.00	28.57	37.50	42.86	50.00	55.36	58.93	69.64
Cohort 1 (mean LoC)	21.0	29.8	44.9	59.9	89.8	119.8	159.8	199.9	260.0	318.3	413.0
Cohort 2 (%)	14.75	18.03	21.31	22.95	31.15	34.43	32.79	37.70	40.98	47.54	50.82
Cohort 2 (mean LoC)	21.0	30.0	45.0	59.8	90.0	120.0	159.9	200.0	259.9	319.2	417.2

The full-pool curve rises with the source-line limit overall, despite the observed cohort 2 decrease from 34.43% at 120 lines to 32.79% at 160 lines. Over the shared rendered-cost range, the Native-anchor control lies above the full-pool selector on both cohorts. On cohort 2, the 420-line full-pool rate is 50.82%, compared with the 68.85% Native operating point and 86.89% gold-aware ceiling. Added execution candidates increase oracle reachability but coincide with lower realized selection under this frozen policy.

(a) Experiment 1 · Crash locality

`astropy-7606 · schematic` `failure_path.py`

```
1 repair_A = normalize(raw)          required A
2 ...  # many executed lines
3 value = consume(repair_A)
4 raise TypeError(value)             crash seed
5 ...  # distant execution
6 repair_B = validate(cfg)           required B
```

Crash seed ≠ complete closure.

Figure 8: Examples for Experiments 1–2. **(a)** A schematic places the crash seed between distant repair locations; the displayed lines are noncontiguous and do not form one execution sequence.

(b) Experiment 2 · Larger pool

matplotlib-25079 · pseudocode — colors.py · autoscale()

```
1 with callbacks.blocked():                    dropped
2   vmin = vmax = None
3   autoscale_None(A)
4 self._changed()                              dropped
5 pool = {1, 2, 3, 4}                          reachable
6 selected = {2, 3}                            incomplete
```

Reachable ≠ selected.

Figure 8: Examples for Experiments 1–2 (continued). **(b)** Pseudocode shows a broader reachable pool whose selected subset remains incomplete. The panels illustrate the recorded failure modes rather than add observations.

G.3 Experiment 3: Local Structural Expansion

The local-structure analysis starts from Native candidates (C0) and adds traceback locations (C1), executed backward definition–use relations (C2), and call predecessors (C3). At the primary $\alpha = 0.90$ setting, each defect's source-line cap is $\lfloor 0.90\,|W_i(N_i)| \rfloor$, where N_i is its Native anchor set. Structured and control candidates pass through the same budgeted selector. The 200 seeded permutations replace additions while matching candidate tier, file, and anchor count under this shared cap; they vary selection within the same defect cohort. Table 19 reports the comparison.

Table 19: Local-structure expansion. Parentheses give reachability and matched denominators, respectively. Native is shown at its original rendered size ($\alpha = 1.00$) as a reference; Expanded and Null mean use the shared $\alpha = 0.90$ cap. Null means use 200 permutations matched on tier, file, and anchor count. The upper-tail p-value counts null statistics at least as large as the observed statistic, with a plus-one correction.

	Unconstrained ceiling		Context-complete rate (%)			
Cohort	**Native**	**Expanded**	**Native ref.**	**Expanded**	**Null mean**	p
Cohort 1 (57 / 56)	35	39	62.50	60.71	61.11	0.990
Cohort 2 (61 / 61)	42	45	68.85	60.66	61.56	1.000

Structural expansion raises closure availability in both cohorts, but its budgeted complete-context rate is 0.40 percentage points below the control mean on cohort 1 and 0.90 points below on cohort 2. These Expanded–Null contrasts share $\alpha = 0.90$; full-budget Native remains a reference. Table 20 separately reports the already-recorded C0 and C3 selectors at the same cap on their original aggregate denominators. C0 uses Native anchors with the same budgeted selector as C3, rather than preserving the entire Native context. These summaries lack paired per-instance outcomes and do not support a significance claim.

Table 20: Existing selectors at $\alpha = 0.90$, read from the frozen aggregate results. Cohort 1's historical 57-row summary includes one missing-Native record assigned zero; the 56-case matched-null comparison in Table 19 excludes it. The missing record is not an observed localization failure.

Cohort	n	**C0 (%)**	**C3 (%)**	**Mean LoC C0 / C3**
Cohort 1	57	57.89	59.65	179.75 / 181.70
Cohort 2	61	60.66	60.66	246.21 / 247.30

Instance-level reconciliation confirms that Experiment 3's valid matched samples are identical to RQ1 and CFRR: 56 defects in cohort 1 and 61 in cohort 2, with empty bidirectional ID differences. The historical 57-row cohort 1 ledger additionally retains `sympy__sympy-21171`. Its canonical gold is nonempty (four lines), but its Native localization record is absent; the recorded exclusion reason is `native_loc_missing`, not invalid source-line counts. The historical mixed

label `file_unreachable_noLOC` does not distinguish these causes. The absence predates this analysis; its generating-run cause remains unknown. Figure 9, view (3), illustrates the gap between supply and selection.

G.4 Experiment 4: Fail–Pass Outcome Labels

The fail–pass analysis tests whether failure-exclusive execution identifies repair code. Let F_i and P_i denote the source lines covered by failing and passing tests, respectively. The experiment uses nine on-path development defects with both coverage sets, fixed before closure scoring. Six of the nine closures satisfy $G_i \cap (F_i \setminus P_i) = \varnothing$: the failing-only set contains no reference repair line. These exploratory cases show that discarding passing-covered code can remove repair-relevant locations on shared execution paths. They test hard set subtraction, not continuous suspiciousness ranking or spectrum-based localization in general. Figure 9, view (4), illustrates this pattern.

(a) Experiment 3 · Local structure

`matplotlib-25498 · pseudocode` — `candidate_builder.py`

```
C0 = native()  # 172
C1 = C0 + traceback()
C2 = C1 + defuse()
C3 = C2 + callers()  # 551        adds gold
closure <= C3                    complete pool
select(C3, budget) -> 2/4        incomplete
```

Larger pool ≠ complete selection.

(b) Experiment 4 · Fail–pass labels

`astropy-14365 · pseudocode` — `spectrum_audit.py`

```
P = run(pass)  # A, B            shared
F = run(fail)  # A, B            shared
only_fail = F - P
gold = {A, B}
gold & only_fail == {}           no gold
# shared path; values differ
```

Fail-only lines miss shared-path repair.

Figure 9: Examples for Experiments 3–4. **(a)** Pseudocode separates closure availability after structural expansion from incomplete budgeted selection. **(b)** Pseudocode places the repair closure on lines shared by failing and passing executions, so the failing-only set omits it. The panels illustrate the recorded patterns rather than add observations.

G.5 Experiment 5: Reversible AST Response

The AST experiment tests whether semantic candidate sites respond more often than matched executed sites. The frozen scope contains 18 valid development instances; 17 have an in-scope non-KEEP site for the paired location comparison. The probe family applies one reversible AST operator to semantic candidates and to executed candidates matched within instance on file, AST type, execution band, and function scale. No-op shams check instrumentation, and every nonzero response chosen for repetition is reproduced. An informative response either propagates to the original failure's state, signature, or location without a new earlier exception (R2), or makes the exact failing test pass (R3). R0 denotes no observable effect, and R1 denotes a coverage-only change; neither counts as informative. For each defect, Q_{sem} and Q_{matched} are the fractions of semantic and matched probes classified R2 or R3.

Averaging the within-defect contrasts gives $\Delta Q = -0.029$ (permutation $p = 0.778$), with no detected response advantage for semantic sites. Figure 10, view (5), shows the measured R0/R2 outcomes in one case. The groups are defined by candidate provenance; behavioral response and

repair-closure membership remain separate endpoints. This small exploratory comparison does not establish response-rate equivalence.

G.6 Experiment 6: Directed Predicate Certification

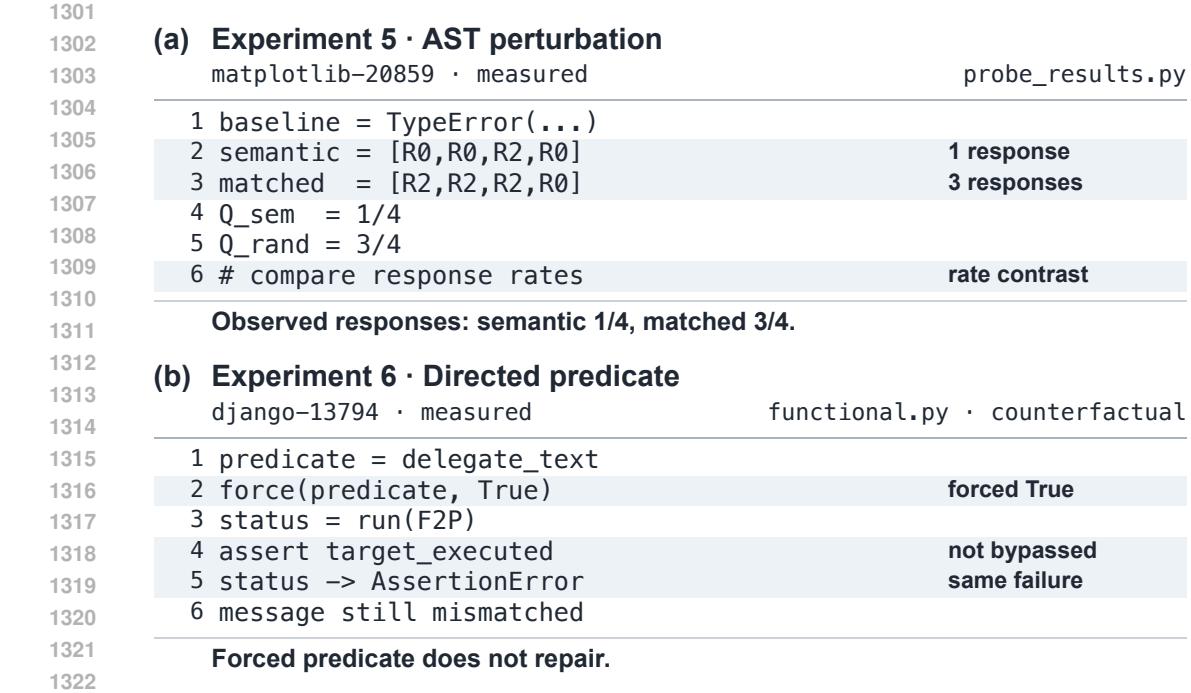


Figure 10: Measured examples for Experiments 5–6. **(a)** One of four semantic probes yields an informative R2 response, whereas three of four matched executed-site probes do; R0 denotes no observable effect, and R2 denotes a change in the original failure's state, signature, or location without an earlier exception. **(b)** Forcing the predicted predicate direction preserves the original failure in one certification case. These panels are individual examples; the aggregate results appear in the surrounding text.

The final dimension tests whether predicted Boolean repair directions can be behaviorally certified. For each resolvable semantic constraint, the frozen experiment forces an existing Boolean predicate in the predicted direction, forces the opposite direction, and applies a semantics-preserving sham. Certification requires the predicted direction to pass repeatedly while still executing the original target, the opposite direction to retain the original failure, and the source to restore cleanly. None of the 32 constraints across 17 valid exact-F2P instances achieves complete certification. No sham is certified, and restoration succeeds in every case. Figure 10, view (6), gives one measured example. These exploratory measurements separate local predicate response from repair-closure composition under the stated certification protocol; the directions within a defect are not independent defect samples and provide no general bound on perturbation-based localization.

G.7 Cross-Dimension Evidence Ledger

Table 21 maps each experiment to its population, recorded outcome, and role in the RQ2 argument.

Table 21: Complete RQ2 evidence ledger. Each row retains its own denominator and trace contract.

Evidence	Population	Recorded result and role
Repeated exact crash test	101 primary defects; 126 eligible	+0.008 percentage points; one improvement, two regressions, and 98 ties.

Continued on next page

Table 21 (continued)

Evidence	Population	Recorded result and role
Action-supply decomposition	101 primary defects	79 zero-increment actions; among 22 nonzero actions, one improvement, two regressions, and 19 ties.
Earlier exact audit	17-defect development intersection	16 execution-feasible and two locked completions; reachability precursor.
Candidate breadth	61 defects in development cohort 2	86.9% gold-aware ceiling versus 42.6% frozen realization under a whole-module proxy.
Local structure	57 defects in cohort 1 (56 matched) and 61 in cohort 2	The structural-expansion ceiling reaches 39 in cohort 1 and 45 in cohort 2; matched differences are -0.40 and -0.90 percentage points.
Further crash-selection audit	Nine valid defects	Two complete closures; $D = 0$, aggregate permutation $p = 0.5821$; supporting crash-selection evidence.
Failing-only contrast	Nine defects with failing/passing coverage	Six of nine closures contain no failing-only gold line.
AST perturbations	18 valid; 17 paired-site usable	$\Delta Q = -0.029$, $p = 0.778$.
Directed predicates	17 defects; 32 constraints	No repeated directional certification with opposite and sham controls.

Candidate expansion exposes closure headroom under the respective trace contracts, while the tested selectors do not consistently realize it. The fail–pass cases show that required lines can lie on shared paths. Experiment 5 detects no semantic-site response advantage, and Experiment 6 obtains no full directional certificate. These distinct observations motivate preserving semantic localization while testing bounded execution-derived additions.

H RQ3: CFRR Diagnostic Details

CFRR uses the same 117-defect development cohort as RQ1. For each defect, N_i is the union of frozen Native anchors. The fixed candidate pool combines Native and vote anchors, related regions, and whole-module proxy coverage within found files; its subsets form the diagnostic action family $\mathcal{A}_i$. The source-line cap is $B_i = |W_i(N_i)|$, and the evaluated gold-blind output is N_i, so $\Pi_i = \{N_i\}$. Native's rendered contexts complete 77 closures at a mean cost of 244.09 unique source lines. After candidate construction is fixed, the gold-aware minimum-union calculation finds 103 reachable closures, each fitting its declared cap. Reachability and fit therefore coincide in this cohort. Brute-force verification confirms the candidate-headroom calculation. Exposure, reachability, capacity fit, and Native realization share the same denominator; evidence from other contracts remains disaggregated.

Retained input evidence. The Native files for development cohorts 1 and 2 contain 56 and 61 records, respectively, each retaining the instance identifier, found files, related locations, and four edit-location samples. The canonical mapper unions anchors across these samples before the single ± 10 expansion; $\Pi_i = \{N_i\}$ therefore contains one anchor union, rendered and scored as a single output rather than four separate alternatives. This development-audit construction is distinct from the one-response-per-stage Native-generation configuration used in the three-model closure evaluation (Appendix N.2).

The visible wrapper requests edit locations around related code in upstream-found files, whereas the audit separately reads vote and whole-module coverage caches. These interfaces establish how the audit pool is assembled, but the retained Native records omit serialized prompts and per-candidate visibility. Current wrapper defaults cannot certify historical request-level settings. The canonical 117-row ledger agrees instance by instance with the retained found-file candidate-pool results for Native cost, reachability, minimum complete cost, capacity fit, and Native completeness; that record agreement supports the diagnostic decomposition without resolving the missing prompt evidence.

H.1 Nested Contracts and Loss Decomposition

For a fixed instance, candidate universe, action family, and budget, the contracts satisfy

$$\mathrm{Real}_i \le \mathrm{Fit}_i \le \mathrm{Reach}_i \le \mathrm{Exp}_i. \tag{11}$$

The nesting follows from set inclusion. Every realized context is a fitting action, every fitting action is reachable, and repair-file exposure is a prerequisite for closure containment. This nesting gives a

first-failure decomposition. Let Z_i record the first failed stage, or COMPLETE when all four checks succeed. Table 22 reports the canonical 117-row accounting:

$$1 - C_i^{\text{Native}} = \mathbf{1}[Z_i = \text{EXPOSURE}] + \mathbf{1}[Z_i = \text{REACHABILITY}] + \mathbf{1}[Z_i = \text{CAPACITY}] + \mathbf{1}[Z_i = \text{REALIZATION}], \tag{12}$$

with exactly one active term for every incomplete defect. The resulting invariant is $13 + 1 + 0 + 26 = 40 = 117 - 77$. The first failed stage indicates which intervention class to examine: retrieval, candidate construction, compression, or gold-blind ranking. Figure 11 visualizes the same mutually exclusive accounting.

Table 22: Canonical CFRR first-state accounting in the two development cohorts. Failure columns are mutually exclusive and sum to the 40 Native-incomplete defects.

Cohort	Complete	Exposure	Reachability	Capacity	Realization	Total
Cohort 1	35	6	0	0	15	56
Cohort 2	42	7	1	0	11	61
Total	77	13	1	0	26	117

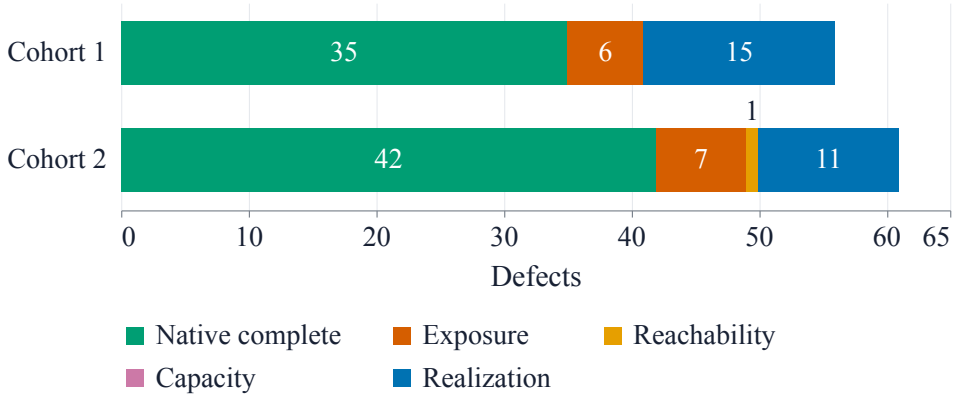


Figure 11: Canonical CFRR first-state ledger. Cohort 1 and cohort 2 contain 56 and 61 defects; each defect occupies exactly one mutually exclusive state, and the bars sum to 117.

The two development-cohort ledgers expose a gap between available complete candidates and Native's selected contexts.

H.2 GRADED DIAGNOSTICS

Binary contracts locate the first failed stage. Graded quantities measure distance within a stage. Table 23 states how each field is computed and which question it answers. For a fixed action family $\mathcal{A}_i$, define

$$r_i^\star = \max_{A \in \mathcal{A}_i} \frac{|G_i \cap W_i(A)|}{|G_i|}, \qquad b_i^\star = \min_{A \in \mathcal{A}_i : G_i \subseteq W_i(A)} \tau_i(A), \tag{13}$$

where $b_i^\star = \infty$ when the closure is unreachable. For a set of realized outputs $\Pi_i \subseteq \mathcal{A}_i$, let $q_i = \max_{A \in \Pi_i} |G_i \cap W_i(A)|/|G_i|$ and define the realization gap $g_i = r_i^\star - q_i$. Here, $r_i^\star$ measures unconstrained reachable repair-line coverage, $b_i^\star$ measures the minimum capacity required for complete containment, and g_i measures the difference between unconstrained reachable coverage and realized coverage. The gap must be interpreted jointly with $b_i^\star$ and the source-line limit B_i: when a complete candidate fits the budget, it can be interpreted as selection-side headroom; otherwise it may also reflect capacity. Each output is rendered and scored separately. In the present ledger, $\Pi_i = \{N_i\}$, so q_i and C_i^{Native} measure partial and complete containment in Native.

In the 117-row ledger, `source_line_limit` records B_i and `minimum_complete_source_lines` records the finite $b_i^\star$ when reachable. Every reachable

closure fits under gold-aware subset selection with the four-sample budget $B_i = |W_i(N_i)|$. With the candidate pool and rendering rule fixed, strict KEEP ($N_i \subseteq A$) implies $W_i(N_i) \subseteq W_i(A)$; imposing $|W_i(A)| \leq |W_i(N_i)|$ therefore forces equality of the rendered sets. Zero-cost anchors cannot complete an incomplete Native context. Reachable but Native-incomplete cases would consequently fail Capacity under this modified contract; the reported ledger retains its original contract. Reducing only the budget to one retained sample's rendered size cannot improve Fit with the pool fixed. Rebuilding the pool from that sample may also change Reachability, so no numerical reallocation is inferred without that separate audit.

Table 23: CFRR measurement record. The fields keep stage mechanisms separate during aggregation.

Field	Range	Computed from	Question answered
Exp_i	$\{0, 1\}$	Upstream files and closure files	Are all required files visible?
Reach_i	$\{0, 1\}$	Frozen action family and closure	Does any generated action contain the closure?
Fit_i	$\{0, 1\}$	Reachable actions and budget	Can a complete action fit?
Real_i	$\{0, 1\}$	Frozen policy output	Did the deployable policy produce one?
$r_i^\star$	$[0, 1]$	Best frozen action, gold-aware audit	How much closure is reachable?
$b_i^\star$	$\mathbb{N} \cup \{\infty\}$	Minimum complete frozen action	What capacity is minimally required?
q_i	$[0, 1]$	Frozen policy outputs	How much closure did the policy deliver?
g_i	$[0, 1]$	$r_i^\star - q_i$	What is the reachability–realization gap?

Measurement value. CFRR connects the first failed stage to a context-construction question. The observed realization gap motivates testing Native-preserving residual completion. TraceFill evaluates this design separately in a larger cohort using one Native base per defect and model.

I TRACEFILL: DETERMINISTIC INSTANTIATION, GUARANTEES, AND FROZEN DESIGN CHOICES

TRACEFILL deterministically allocates residual capacity. For each eligible defect–model cell, it takes the Native anchor set K_{im}, two qualifying runs of the recorded declared-test input, and three added-source-line caps. Each valid Python source line observed in both runs becomes a candidate anchor. Candidates are ordered by provenance tier—repeat-consistent exception site, call/return endpoint, then other executed line—followed by Native found-file rank, file path, and line number. Starting from KEEP, TraceFill processes the caps in ascending order, retaining accepted anchors and rescanning the candidate order as capacity grows. If no addition fits, it returns KEEP.

The action inventory, three caps, tiered ordering, and deterministic tie breaks are fixed before gold loading. The isolated scorer verifies the generation freeze, action manifest, action-bundle hashes, and its own hash before reconstructing rendered lines from frozen anchors. Same-file, anchor-count, and Exact-LoC controls share KEEP and the added-line cap; infeasible matches remain in the audit ledger.

Frozen Native input. Each action bundle records `base_anchors`, Native rendered cost, found files, and the Native-action hash; this single base is shared across all three caps and all arms for that defect–model cell. All 2,289 endpoint rows bind to a bundle by hash with matching instance and model.

Native construction and availability. The archived Native generator for closure evaluation uses one response per reached stage in a three-stage file/symbol/edit-location pipeline. It submits temperature zero and a 300-token output parameter for each of the three model aliases; these are submitted settings, not a verified common effective decoding cap. Localization-output parsing failures receive an empty edit set under one uniform rule. Appendix N.2 gives the actual request coordinates, empty-base rates, rendered-size summary, and the unresolved multiple-test-selector detail.

Recorded matching contracts. These restricted negative controls preserve Native and are generated separately at each cap. Fixed seeds determine clustered candidate pools and their order; Native and

stable-trace anchors are excluded from these pools. This restriction concerns anchors, not rendered windows, which may overlap executed lines. Same-file selects within the files containing TraceFill's additions at that cap and matches its final rendered source-line count. Anchor-count draws from repository files, near Native anchors or seeded centers, and matches the number of added anchors under the residual cap. Exact-LoC draws from repository files and packs windows to match the final rendered source-line count. Scored equality and availability across all three caps determine the supporting subsets in Appendix J; seeds, selected additions, and infeasible matches remain recorded.

Budget and comparisons. For cell (i, m, b), $B_{imb} = \tau_i(K_{im}) + b$ is the total rendered source-line limit, and $b \in \{51, 170, 414\}$ is the residual cap beyond Native. Table 24 specifies initialization, the frozen tiered-priority scan, budget-specific allocation, and fit-or-skip decisions. KEEP preservation and budget safety are documented below. Comparator-specific complete-case controls compare trace-selected content with restricted generic additions; they do not isolate provenance ordering or establish superiority over unrestricted sampling. Table 27 additionally compares all control arms on 1,965 common pairs. The retrospective analyses in Table 4 vary KEEP, priority, and allocation after reproducing the frozen TraceFill actions. The repeat-consistency analysis measures containment under alternative trace-set choices.

I.1 Deterministic Procedure

Let $E_i^{(1)}$ and $E_i^{(2)}$ be the source lines observed in two qualifying runs of the recorded declared-test input. For an eligible cell, $T_i = E_i^{(1)} \cap E_i^{(2)}$. Initialize $A = K_{im}$ and $L = W_i(A)$ once. Table 24 then processes $b = 51, 170, 414$ in order, retaining accepted anchors between caps.

Table 24: TraceFill generation for one defect–model cell across three residual caps.

Step	State update	Auditable condition
1	$T_i \leftarrow E_i^{(1)} \cap E_i^{(2)}$	Retain valid Python source-line anchors present in both qualifying declared-test runs.
2	Sort T_i	Order by provenance tier, Native file rank, file path, and line number.
3	For each $b \in \{51, 170, 414\}$	Set $B_{imb} = \lvert W_i(K_{im})\rvert + b$; retain the current A and L.
4	Scan each $c \in T_i$	Skip $c \in A$ and otherwise follow the same frozen order at every cap.
5	Compute marginal cost	Set $d_c = \lvert W_i(A \cup \{c\}) \setminus L\rvert$ after one-time ± 10 expansion.
6	Fit or skip	If $\lvert L\rvert + d_c \le B_{imb}$, set $A \leftarrow A \cup \{c\}$ and $L \leftarrow W_i(A)$; otherwise continue.
7	Record the cap	Store the accepted anchors and rendered cost for (i, m, b), then repeat steps 3–7 at the next cap.
8	Freeze the bundle	Store the three budget-specific outputs and the whole action-bundle hash before scoring.

Complexity. Let $n_T = |T_i|$ be the number of stable candidate anchors, u the number of unique rendered lines, and $w \le 21$ the clipped window width. Sorting costs $O(n_T \log n_T)$. Across three fixed caps, set-based marginal-cost updates cost $O(n_T w + u)$, giving $O(n_T \log n_T + n_T w + u)$ time and $O(n_T + u)$ memory.

I.2 Safety Properties

KEEP preservation and containment monotonicity. The procedure initializes with $A = K_{im}$ and only adds anchors, preserving every source line and complete closure in Native. Accepted anchors also carry forward as the cap grows, so $W_i(A_{im,51}) \subseteq W_i(A_{im,170}) \subseteq W_i(A_{im,414})$. Complete containment is therefore nondecreasing across caps. This nesting is verified in the frozen outputs and makes Complete@3 equal to Complete(414) for Native and TraceFill. Controls retain their separately computed Complete@3 scores.

Budget safety. For residual cap b, let $B_{imb} = |W_i(K_{im})| + b$. With $L = W_i(A)$, the marginal-cost test in Step 6 is $|L| + d_c = |W_i(A \cup \{c\})| \le B_{imb}$; a candidate is accepted only if this test holds. Hence every recorded output satisfies $|W_i(A_{imb})| \le B_{imb}$.

Gold blindness. Candidate extraction and ordering use only Native anchors, repeated-execution provenance, and frozen tie breaks; the residual cap governs admission under this fixed ranking. After verifying the generation records, the isolated scorer loads the reference closure and scores the outputs. Table 25 summarizes the packaged fields used for offline reaggregation.

Table 25: TraceFill fields packaged for offline reaggregation and verification.

Field	Granularity	Purpose
Identity and configuration	Action bundle	Records dataset, instance, model, seed, schema, builder version, budgets, expansion rule, status, and `gold_read=false`.
Native and trace bindings	Action bundle	Records Native anchors, found files, and Native rendered cost, plus Native-action, trace-artifact, and trace-summary hashes.
Frozen action arms	Budget-specific arm	Records each arm's added anchors, counts, rendered incremental cost, availability, and KEEP-union flag.
Score provenance	Model–defect pair	Binds the source score record, action archive, closure-score ledger, score index, and reuse map used for reaggregation.
Offline reaggregation	Processed records	Recomputes reported summaries from the fixed action and score ledgers.

J RQ4: Robustness, Controls, and Attrition

The funnel begins with 1,100 logical rows: 1,097 are owner-valid, 852 have repeat-consistent traces, and 821 are closure-evaluable. Verified, Live Lite, and Lite contribute 355, 271, and 195 closure-evaluable rows. Applying the frozen reuse map to these 821 rows yields 763 unique defects. The trace gate excludes 245 rows for unstable or invalid evidence; closure scoring excludes 31 empty canonical Python closures. Table 26 gives the ledger, and Figure 12 shows the funnel. At the physical-instance level, 789 instances have matching exceptions across the two runs, 190 have different exceptions, 24 have invalid trace pairs, and one instance with conflicting stored and recomputed status labels is excluded from trace qualification. All gates use explicit attrition codes.

Table 26: Detailed logical-row attrition. Unique physical counts are 1,004 after cross-benchmark deduplication, 789 among stable rows, and 763 among final closure-evaluable rows.

Benchmark	Raw	Owner-valid	Stable	Empty gold	Final
SWE-bench Verified	500	500	361	6	355
SWE-bench Live Lite	300	297	288	17	271
SWE-bench Lite	300	300	203	8	195
Total logical rows	1,100	1,097	852	31	821

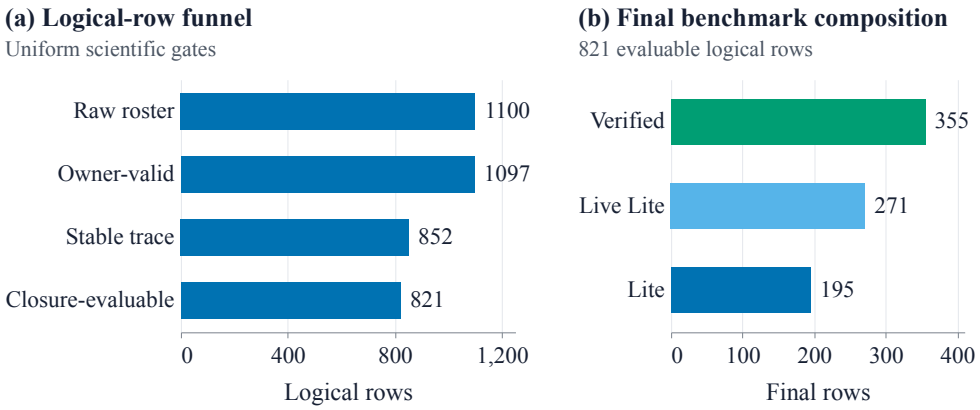


Figure 12: RQ4 logical-row funnel and final benchmark composition. Cross-benchmark deduplication yields 763 unique final instances.

J.1 Budget Sensitivity and Descriptive Cells

The displays in this subsection answer different robustness questions. Table 27 summarizes the fixed panel, common-control cohort, and file-scope sensitivity, while Table 29 provides all 135 method–benchmark–model–budget cells with their eligible denominators. Table 30 isolates the pooled response to residual capacity. Table 31 and Figure 13 show benchmark–model heterogeneity at $b = 414$, and Table 32 reports that heterogeneity across all three caps. Together they supplement the main-text matrix in Table 3 without changing its endpoint or denominator.

Table 27: Complete(414) results and same-model COMPLETE@3 common-cohort sensitivity.

Model-stratified primary result				
Model stratum	**Pairs**	**Native (%)**	**TraceFill (%)**	**Gain (points)**
DeepSeek-V3.2	763	16.51	24.77	+8.26
Qwen3.5-Plus	763	52.29	58.85	+6.55
GPT-5.3-Codex	763	31.32	40.10	+8.78
Fixed panel	**2,289**	**33.38**	**41.24**	**+7.86**
Common-cohort COMPLETE@3 supporting controls				
Comparator	**Pairs**	**Comparator (%)**	**TraceFill (%)**	**Gain (points)**
Native	1,965	32.37	40.10	+7.74
Same-file	1,965	34.71	40.10	+5.39
Anchor-count	1,965	32.57	40.10	+7.53
Exact-LoC	1,965	32.77	40.10	+7.33
Same common cohort, Complete(414)				
Native	1,965	32.37	40.10	+7.74
Same-file	1,965	33.84	40.10	+6.26
Anchor-count	1,965	32.57	40.10	+7.53
Exact-LoC	1,965	32.57	40.10	+7.53
File-scope sensitivity				
File scope	**Pairs**	**Native (%)**	**TraceFill (%)**	**Gain (points)**
Single-file	1,824	40.90	50.44	+9.54
Cross-file	465	3.87	5.16	+1.29

Note. The common cohort is post-hoc. Table 29 gives all 135 cells and denominators; Appendix J gives eligibility rules, paired directions, and clustered uncertainty.

Single-cap versus any-budget controls. On the same 1,965 pairs, TraceFill completes 788 contexts at both endpoints. Same-file completes 665 at $b = 414$ and 682 under COMPLETE@3; Anchor-count completes 640 at both; Exact-LoC completes 640 and 644. Thus, the three-budget envelope adds 17, zero, and four control successes. The post-hoc Complete(414) gains over these controls are $+6.26$, $+7.53$, and $+7.53$ points, with repository-clustered 95% intervals $[4.38, 9.11]$, $[5.57, 10.74]$, and $[5.52, 10.85]$; paired wins/losses are 139/16, 150/2, and 152/4. Exact two-sided repository sign-flip p-values are 4.14×10^{-8}, 4.66×10^{-10}, and 1.47×10^{-8}. These marginal analyses retain frozen control realizations, use 10,000 draws with the recorded RQ4 seed derivation and explicit cap-414 labels, and make no simultaneous-inference claim. They are not expectations over new control seeds.

Completeness gains from nonempty Native contexts. A post-hoc split uses each localization model's own Native rendered size, rather than the shared DeepSeek repair inputs. Among nonempty bases, Native and TraceFill complete $126 \rightarrow 138$ of 201 DeepSeek cases, $399 \rightarrow 449$ of 755 Qwen cases, and $239 \rightarrow 278$ of 592 GPT cases, with zero Native-paired losses. These strata account for 101 of the 180 additional complete model–defect pairs; empty bases account for the remaining 79 ($51/562$, $0/8$, and $28/171$ for DeepSeek, Qwen, and GPT). Nonempty therefore denotes available source context, not necessarily correct localization.

Table 28: Post-hoc Complete@3 on nonempty Native bases. Each comparator uses its original arm-specific availability and matching rules; intervals are marginal 95% repository-bootstrap intervals for TraceFill minus comparator.

Model	Comparator	n	Comparator (%)	TraceFill (%)	Δ (pp)	95% CI
DeepSeek	Native	201	62.69	68.66	+5.97	[2.59, 11.05]
DeepSeek	Same-file	198	66.16	68.69	+2.53	[-1.56, 7.33]
DeepSeek	Anchor-count	199	63.32	68.84	+5.53	[2.12, 10.73]
DeepSeek	Exact-LoC	169	62.72	69.23	+6.51	[2.48, 12.50]
Qwen	Native	755	52.85	59.47	+6.62	[4.79, 9.59]
Qwen	Same-file	748	55.35	59.63	+4.28	[2.72, 6.46]
Qwen	Anchor-count	746	52.82	59.38	+6.57	[4.76, 9.39]
Qwen	Exact-LoC	662	52.27	58.46	+6.19	[4.39, 9.28]
GPT	Native	592	40.37	46.96	+6.59	[4.29, 9.59]
GPT	Same-file	587	42.25	47.19	+4.94	[2.48, 7.77]
GPT	Anchor-count	584	40.41	46.75	+6.34	[4.07, 9.34]
GPT	Exact-LoC	522	39.08	45.98	+6.90	[4.35, 10.29]

The intervals use 10,000 repository-clustered draws with deterministic model–stratum–comparator seeds derived from base seed 2026090301. They condition on frozen outcomes and are not simulta-

neous or selection-adjusted intervals. DeepSeek's Same-file interval includes zero, so the control advantages are not uniformly resolved across model strata. The eight empty Qwen bases contain no observed complete output; this small subgroup does not establish equivalence. These are containment results, not downstream repair effects.

Table 29: Complete(b) for all 135 frozen method–benchmark–model–budget cells. Rates are percentages; n is the eligible logical-cell count for the method within that benchmark–model stratum. Controls are post-hoc supporting analyses, and Exact-LoC is a sensitivity.

Benchmark	**Model**	**Method**	n	$b = 51$	$b = 170$	$b = 414$
Verified	DeepSeek-V3.2	Native	355	18.03	18.03	18.03
Verified	DeepSeek-V3.2	TraceFill	355	19.44	21.97	26.76
Verified	DeepSeek-V3.2	Same-file	353	18.41	18.13	19.83
Verified	DeepSeek-V3.2	Anchor-count	355	18.03	18.03	18.03
Verified	DeepSeek-V3.2	Exact-LoC	299	17.73	17.73	17.73
Verified	Qwen3.5-Plus	Native	355	60.28	60.28	60.28
Verified	Qwen3.5-Plus	TraceFill	355	62.82	63.38	65.07
Verified	Qwen3.5-Plus	Same-file	353	61.19	60.62	61.19
Verified	Qwen3.5-Plus	Anchor-count	354	60.17	60.17	60.17
Verified	Qwen3.5-Plus	Exact-LoC	298	59.40	59.73	59.73
Verified	GPT-5.3-Codex	Native	355	27.89	27.89	27.89
Verified	GPT-5.3-Codex	TraceFill	355	29.86	32.39	37.46
Verified	GPT-5.3-Codex	Same-file	354	28.25	28.25	28.25
Verified	GPT-5.3-Codex	Anchor-count	354	27.97	27.97	27.97
Verified	GPT-5.3-Codex	Exact-LoC	300	26.00	26.33	26.00
Live Lite	DeepSeek-V3.2	Native	271	11.81	11.81	11.81
Live Lite	DeepSeek-V3.2	TraceFill	271	14.76	16.97	18.45
Live Lite	DeepSeek-V3.2	Same-file	265	12.08	12.45	13.58
Live Lite	DeepSeek-V3.2	Anchor-count	269	11.90	11.90	12.64
Live Lite	DeepSeek-V3.2	Exact-LoC	257	11.67	12.06	12.06
Live Lite	Qwen3.5-Plus	Native	271	34.69	34.69	34.69
Live Lite	Qwen3.5-Plus	TraceFill	271	37.64	39.85	41.70
Live Lite	Qwen3.5-Plus	Same-file	266	36.47	35.34	36.47
Live Lite	Qwen3.5-Plus	Anchor-count	266	34.21	34.21	34.59
Live Lite	Qwen3.5-Plus	Exact-LoC	261	34.48	34.48	34.87
Live Lite	GPT-5.3-Codex	Native	271	32.84	32.84	32.84
Live Lite	GPT-5.3-Codex	TraceFill	271	35.06	38.38	40.59
Live Lite	GPT-5.3-Codex	Same-file	267	34.08	33.71	35.21
Live Lite	GPT-5.3-Codex	Anchor-count	262	32.44	32.44	32.44
Live Lite	GPT-5.3-Codex	Exact-LoC	258	33.33	33.33	33.72
Lite	DeepSeek-V3.2	Native	195	23.08	23.08	23.08
Lite	DeepSeek-V3.2	TraceFill	195	26.15	28.72	32.82
Lite	DeepSeek-V3.2	Same-file	194	23.71	23.71	23.71
Lite	DeepSeek-V3.2	Anchor-count	194	23.20	23.20	23.20
Lite	DeepSeek-V3.2	Exact-LoC	154	21.43	22.08	21.43
Lite	Qwen3.5-Plus	Native	195	68.72	68.72	68.72
Lite	Qwen3.5-Plus	TraceFill	195	71.79	73.85	76.41
Lite	Qwen3.5-Plus	Same-file	195	68.72	69.23	69.74
Lite	Qwen3.5-Plus	Anchor-count	192	68.75	68.75	68.75
Lite	Qwen3.5-Plus	Exact-LoC	157	71.34	71.34	71.34
Lite	GPT-5.3-Codex	Native	195	35.38	35.38	35.38
Lite	GPT-5.3-Codex	TraceFill	195	38.46	41.03	44.62
Lite	GPT-5.3-Codex	Same-file	195	35.90	36.92	35.38
Lite	GPT-5.3-Codex	Anchor-count	195	35.38	35.38	35.90
Lite	GPT-5.3-Codex	Exact-LoC	154	34.42	34.42	34.42

Table 30: Single-cap Complete over 763 defects and three separate models (2,289 descriptive panel rows). The final row reports COMPLETE@3, aggregating containment scores across the three budgets within each model. W/L/T denotes improved, regressed, and tied model–defect pairs.

Residual cap b	Native	TraceFill	Δ (pp)	Relative gain	W/L/T
51	33.38%	35.91%	+2.53	+7.59%	58/0/2,231
170	33.38%	38.18%	+4.81	+14.40%	110/0/2,179
414	33.38%	41.24%	+7.86	+23.56%	180/0/2,109
COMPLETE@3	33.38%	41.24%	+7.86	+23.56%	180/0/2,109

TraceFill's single-cap Complete increases with residual capacity, with zero paired losses relative to Native. For Native and TraceFill, all successes at 51 or 170 also succeed at 414. Consequently, COMPLETE@3 equals Complete(414) empirically in these evaluated outputs.

Table 31: All nine 414-line Complete(b) cells. Counts are Native → TraceFill over the benchmark-specific evaluable denominator; W/L/T denotes improved, regressed, and tied model–defect pairs. Verified–Lite aliases retain both logical memberships; pooled results use the reuse map. The 414 entries empirically equal the corresponding COMPLETE@3 cells for Native and TraceFill.

Benchmark	Model	n	Complete contexts	Complete rate (%)	Δ (pp)	W/L/T
Verified	DeepSeek-V3.2	355	64→95	18.03→26.76%	+8.73	31/0/324
Verified	Qwen3.5-Plus	355	214→231	60.28→65.07%	+4.79	17/0/338
Verified	GPT-5.3-Codex	355	99→133	27.89→37.46%	+9.58	34/0/321
Live Lite	DeepSeek-V3.2	271	32→50	11.81→18.45%	+6.64	18/0/253
Live Lite	Qwen3.5-Plus	271	94→113	34.69→41.70%	+7.01	19/0/252
Live Lite	GPT-5.3-Codex	271	89→110	32.84→40.59%	+7.75	21/0/250
Lite	DeepSeek-V3.2	195	45→64	23.08→32.82%	+9.74	19/0/176
Lite	Qwen3.5-Plus	195	134→149	68.72→76.41%	+7.69	15/0/180
Lite	GPT-5.3-Codex	195	69→87	35.38→44.62%	+9.23	18/0/177

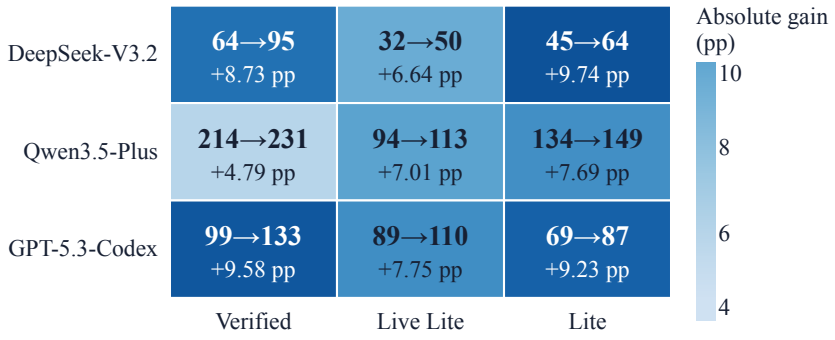


Figure 13: Absolute Complete(414) gains across all benchmark–model cells; text gives Native→TraceFill complete-context counts.

Repository-clustered 95% confidence intervals for the COMPLETE@3 changes, in percentage points, are [5.60, 12.30] for DeepSeek-V3.2, [4.77, 9.45] for Qwen3.5-Plus, and [6.24, 12.52] for GPT-5.3-Codex; the fixed-panel and aligned-control intervals appear in Table 34.

Development-overlap and file-scope sensitivities. A post-hoc sensitivity removes 83 defects shared with the repeatedly used development cohorts, leaving 680 defects and 2,040 model–defect pairs. In this subset, COMPLETE@3 successes increase from 654 to 798, a gain of 7.06 percentage points with a repository-clustered 95% CI of [4.87, 10.28]. In a post-hoc scope split, the 608 single-file defects contribute 1,824 pairs, among which successes increase from 746 to 920, a gain of 9.54 percentage points (95% CI [6.72, 14.33]). For the 155 cross-file defects (465 pairs), the corresponding count changes from 18 to 24, a gain of 1.29 percentage points (95% CI [0.22, 2.52]). Only four cross-file repositories contain discordant pairs, giving block sign-flip $p = .125$.

Resampling configuration and membership. The containment intervals above use 10,000 whole-repository bootstrap draws, retaining each sampled repository's included defect–model pairs. Each draw recomputes the pooled paired percentage-point difference using its sampled row count; the interval takes linearly interpolated empirical percentiles at 0.025 and 0.975. Model-stratified comparisons each use 72 repositories. Comparator-specific Same-file, Anchor-count, and Exact-LoC intervals use 71, 72, and 71 repositories; development-overlap exclusion retains 72, and single-file and cross-file sensitivities use 61 and 51. Verified–Lite aliases contribute once. The common 1,965-pair subset contains 683 defects across 70 repositories: 630 defects retain three model rows, 22 retain two, and 31 retain one, totaling 659 DeepSeek rows and 653 each for GPT and Qwen. Availability and matching requirements apply across all three budgets before resampling; unavailable control rows are excluded without imputation. These intervals condition on frozen outputs, trace pools, and control realizations; bootstrap draws do not regenerate them or resample the control seeds.

Who is excluded by control matching? From 2,289 pairs, Same-file, Anchor-count, and Exact-LoC retain 2,268, 2,267, and 2,001; the common intersection retains 1,965. Same-file excludes 17 pairs unavailable at one or more caps and four available at all caps but failing scored-size parity. Anchor-count excludes 22 unavailable pairs and does not require LoC parity. Exact-LoC excludes two unavailable pairs and 286 available at all caps but failing scored-size parity. The 288 Exact-LoC exclusions have larger mean Native contexts than retained pairs (377.19 versus 212.94 LoC), similar empty-base proportions (32.29% versus 32.38%), and higher observed Native/TraceFill completeness (39.24/47.92% versus 32.53/40.28%). No valid matched-budget contrast is inferred for excluded pairs; most have observed outputs that fail the cost contract. These post-hoc characteristics do not establish missing-at-random or the direction of bias in a valid paired contrast; the aligned common-cohort comparison avoids comparing rates on different members.

Stored statistical seeds. The main interval uses seed 20260829, and development-overlap exclusion uses 20260902. Other RQ4 seeds are deterministically derived from the first 16 hexadecimal digits of SHA-256 applied to `2026090301|label`. The seeds, row counts, and intervals are retained in `reports/complete_at_3_budget_portfolio.json`; `scripts/derive_complete_at_3.py` defines the exact seed labels and implements the grouping and percentile rules. These bootstrap seeds differ from the frozen control-construction seed 20260817. Both files are relative to the artifact root.

Table 32: All 27 single-cap Complete(b) results in nine descriptive benchmark–model cells. Native is invariant across caps; each cap reports the TraceFill complete count and its percentage-point gain over Native. Verified–Lite aliases retain both logical memberships; pooled results use the reuse map. The 414 entries empirically equal the corresponding COMPLETE@3 cells for Native and TraceFill.

Benchmark	Model	n	Native	$+51$ count / gain	$+170$ count / gain	$+414$ count / gain
Verified	DeepSeek-V3.2	355	64	69 / +1.41	78 / +3.94	95 / +8.73
Verified	Qwen3.5-Plus	355	214	223 / +2.54	225 / +3.10	231 / +4.79
Verified	GPT-5.3-Codex	355	99	106 / +1.97	115 / +4.51	133 / +9.58
Live Lite	DeepSeek-V3.2	271	32	40 / +2.95	46 / +5.17	50 / +6.64
Live Lite	Qwen3.5-Plus	271	94	102 / +2.95	108 / +5.17	113 / +7.01
Live Lite	GPT-5.3-Codex	271	89	95 / +2.21	104 / +5.54	110 / +7.75
Lite	DeepSeek-V3.2	195	45	51 / +3.08	56 / +5.64	64 / +9.74
Lite	Qwen3.5-Plus	195	134	140 / +3.08	144 / +5.13	149 / +7.69
Lite	GPT-5.3-Codex	195	69	75 / +3.08	80 / +5.64	87 / +9.23

J.2 OFFICIAL-HARNESS ENDPOINT AND METRIC IDENTITY

All three generators apply the same Native-preserving TRACEFILL rule to empty and nonempty bases from frozen DeepSeek-V3.2 anchors. The renderer expands each anchor once by ± 10 lines, merges overlaps, and emits numbered source regions. Rows failing the trace-stability gate use identical empty source contexts in both arms.

Meaning of the gate-failure fallback. In the recorded upstream procedure, all 215 physical trace-gate failures are excluded before Native stage-one generation. The repair renderer instantiates uniform

empty source bytes for both arms on these cases. Thus, the Native fallback described in Section 5.3 refers to the recorded request-side input; it does not establish what an independently generated pre-gate Native context would contain.

DeepSeek's frozen repair requests use temperature 0.2 and a 4,096-token output cap.

Repair endpoint totals. With 1,004 physical defects per model, the recorded resolved counts are $52 \to 69$ for Qwen, $68 \to 84$ for GPT, and $20 \to 26$ for DeepSeek: respectively $+1.69$, $+1.59$, and $+0.60$ percentage points, or $+32.7\%$, $+23.5\%$, and $+30.0\%$ relative to their own Native counts. The closure analysis instead uses each model's own Native base; it is not a one-to-one evaluation of those same localization–repair pipelines and does not identify closure improvement as the cause of repair gains.

Each arm requests one repair per dataset–instance–model pair, reusing the response when requests are byte-identical. A successful request yields one patch; pair-symmetric technical failures receive zero in the full-roster endpoint. Figure 14 shows the nine cell-level differences, and Table 33 gives the technical-availability sensitivity. Under this protocol,

$$\text{Resolved}_i = \text{Pass@1}_i = \text{BehavioralCorrect}_i \tag{14}$$

for every scored row: when a patch is available, the sole patch passes the official harness exactly when the instance is resolved. The terms therefore describe one binary observation and share a numerator.

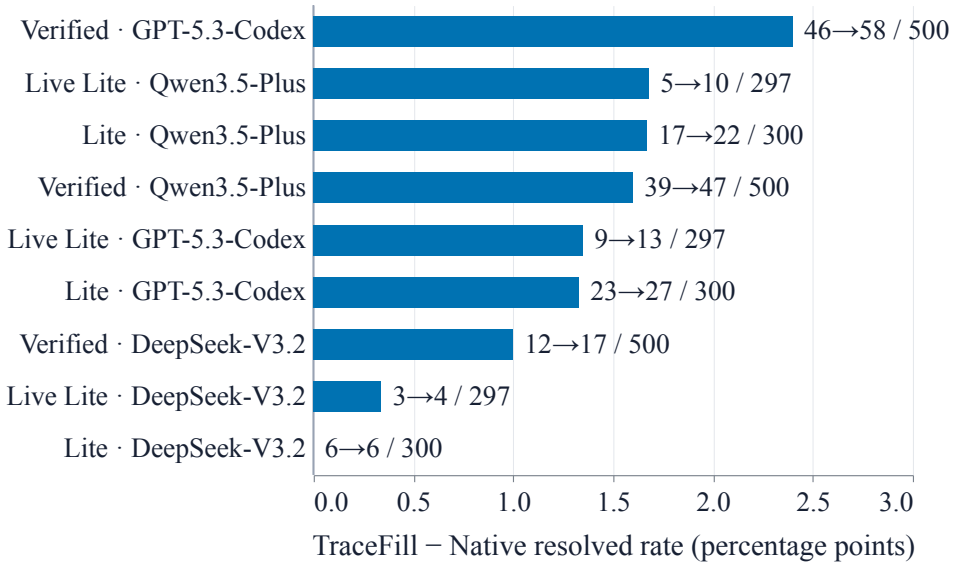


Figure 14: Official resolved-rate differences for the nine benchmark–model cells; labels give Native and TraceFill success counts alongside the roster size.

Repair arms are paired by dataset, instance, and model, with outcomes defined by the official harness. Eight benchmark–model cells improve and one is unchanged. Closure containment and correct-patch success remain separate endpoints.

Table 33: Complete-case sensitivity for technical availability. The full-roster descriptive analysis assigns pair-symmetric technical failures an outcome of zero, so resolved numerators are unchanged.

Model	Technically available pairs	Native resolved	TraceFill resolved	Δ (pp)
DeepSeek-V3.2	887	20	26	+0.68
Qwen3.5-Plus	937	52	69	+1.81
GPT-5.3-Codex	1,003	68	84	+1.60

Aligned-cohort and technical sensitivities. Restricting repair to the 763 deduplicated closure-evaluable instances per model, TRACEFILL resolves 22, 58, and 68 cases for DeepSeek-V3.2, Qwen3.5-Plus, and GPT-5.3-Codex, compared with 16, 41, and 52 for Native, respectively. This reaggregation reports repair on the same defect cohort used for the containment analysis. For the separate full-roster technical-availability sensitivity, complete-case denominators are 887, 937, and 1,003; model-specific resolved numerators and discordant counts are unchanged.

J.3 FOLLOW-UP REPAIR STRATIFICATION AND AUDIT

Post-hoc stratification and uncertainty. Stratification uses the shared DeepSeek Native source-code field, not each generator's localization output from the containment evaluation. For each model and stratum, we report Native-only and policy-only successes and their net difference. We compute 20,000 paired-defect bootstrap draws and, separately, 20,000 whole-repository bootstrap draws, using a base seed of 20260918 and deterministic model–stratum seed derivation. Each repository draw resamples repositories with replacement and recomputes the difference using the sampled number of defects. Intervals are the 2.5th and 97.5th percentiles.

The unadjusted two-sided exact McNemar p-values for the three full-roster comparisons are 0.03125, 0.01151, and 0.02263 for DeepSeek, Qwen, and GPT. The audit separately applies Holm adjustment to the six model-by-Native-availability comparisons within the trace-stable cohort. The exact tests use a defect-independence approximation, while repository resampling supplies a separate clustering sensitivity.

Response audit of the repair Native base. The shared DeepSeek repair base contains 206 nonempty and 583 empty contexts among 789 trace-stable defects. The empty contexts comprise 581 stage-one parsing failures, one stage-two parsing failure, and one empty stage-three context. All stage-one response contents are nonempty. Among the 581 stage-one failures, 567 responses contain an exact Python path present in the frozen repository structure; all 567 contain valid paths wrapped in inline backticks, and 560 also contain list numbering or bullets. The recorded full-line path matching rejects these formats. Length termination co-occurs with all 581 failures, but 206 other length-terminated responses pass stage-one parsing. The audit therefore distinguishes nonempty responses, parsing acceptance, and rendered context availability; it does not isolate a truncation-only cause or measure the counterfactual performance of a corrected parser.

Across the full repair roster, 798 of 1,004 recorded Native source contexts are empty (79.48%): 583 trace-stable cases and 215 cases excluded before Native generation by the trace gate. This source-context count does not imply an empty prompt, an empty model response, or a failed repair outcome.

The 763-defect closure panel and the 789-defect trace-stable repair panel have different eligibility rules. The repair-specific response audit above uses the latter panel; its counts must not be substituted for the canonical counts in Table 42.

Scope of the follow-up repair audit. The audit verified the byte sizes and SHA-256 hashes of all 2,008 frozen source-context files and independently reproduced the stratified summaries from the released repair ledger. It also replayed the available Native parsing records. These checks establish frozen-file identity and offline reaggregation. The complete original repair request, response, and harness-output chain was not recovered, so the audit does not independently authenticate every source byte in the HTTP requests actually sent or re-execute the repair evaluation.

J.4 STRUCTURAL CONTROLS ACROSS THE THREE BUDGETS

The three structural controls preserve the KEEP base and replace TraceFill's content-selection rule. We compare all arms using COMPLETE@3, with comparator availability evaluated across all three caps. At each cap, Same-file additions are restricted to files containing TraceFill's accepted additions and require scored final-LoC equality. Anchor-count matching fixes the number of added anchors under the same residual cap; the post-hoc Exact-LoC subset requires scored final-LoC equality at every cap. Table 34 reports the paired directions and method-specific denominators.

Table 34: Aligned post-hoc supporting controls under the same-model three-budget endpoint. Each method-specific denominator requires that comparator to be available at all three budgets. Same-file and Exact-LoC rows additionally require scored rendered-cost equality at every cap; Exact-LoC is sensitivity evidence.

Comparator	Paired n	W/L/T	Δ (pp)	95% CI	Role matched
Native/KEEP	2,289	180/0/2,109	+7.86	[5.80, 11.00]	KEEP baseline
Same-file additions	2,268	156/26/2,086	+5.73	[3.76, 8.40]	Files + LoC
Anchor-count matched	2,267	176/2/2,089	+7.68	[5.59, 10.76]	Anchor count
Exact-LoC subset	2,001	154/7/1,840	+7.35	[5.33, 10.55]	Rendered LoC

W/L/T denotes TraceFill-only complete, comparator-only complete, and unchanged outcomes. Intervals use repository-clustered resampling.

A post-hoc sensitivity restricts Native, Same-file, Anchor-count, and Exact-LoC to their common set of 1,965 model–defect pairs. On these pairs, TRACEFILL has advantages of 7.74, 5.39, 7.53, and 7.33 percentage points, respectively.

TRACEFILL retains higher containment than these post-hoc comparators. The Exact-LoC sensitivity uses the 2,001 pairs satisfying its matching contract at all three caps. The trace-set alternatives all yield 41.24% COMPLETE@3 on the retained closure cohort and are evaluated only after the original two-run qualification gate. They therefore do not measure a single-run admission policy on the full roster. The intersection removes 579 run-specific lines across 789 qualifying physical defects. The gate excludes 215/1,004 physical defects, corresponding to 245/1,097 logical rows: 190 physical cases have different exceptions, 24 have invalid traces, and one has conflicting status labels (Appendix J). Collecting two traces requires an additional execution relative to a single-trace protocol, but end-to-end latency and total cost were not measured. Single-trace collection remains a potential lower-execution alternative whose applicability and repair outcomes require separate evaluation.

J.5 CROSS-ENDPOINT INTERPRETATION

At residual cap $b = 414$, every closure benchmark–model cell improves by 4.79 to 9.74 points with zero paired losses. In the descriptive official-harness matrix, eight cells improve and one is unchanged, with differences from 0.00 to 2.40 points. Containment and repair use overlapping cohorts with different eligibility rules. Closure scoring requires repeat-consistent traces and nonempty canonical Python closures, yielding 763 defects and 2,289 model–defect pairs. Repair scoring retains all 1,004 deduplicated instances per model under one request per arm. At the repair stage, 215 instances fail the trace-stability gate and share the Native request across arms; the corresponding closure gate excludes 245 of 1,097 logical rows. Table 35 joins the ledgers by defect and model and retains examples of both repair improvement and regression. These cases compare aligned measurements; they do not isolate the effect of completing the repair input.

Table 35: Post-hoc ledger cases aligned by defect and model. Within each stratum, we select the first row ordered by SHA256(`instance_id` ‖ `model`).

Instance / model	Selection stratum	Closure recall	Repair outcome	Recorded contrast
`sphinx-7889` / DeepSeek-V3.2	Closure gain; both unresolved	0% → 100%	Unresolved → Unresolved	Repair remains unresolved in both arms.
`seaborn-3010` / GPT-5.3-Codex	Both complete; TraceFill-only success	100% → 100%	Unresolved → Resolved	Equal closure recall; repair improves.
`xarray-6461` / Qwen3.5-Plus	Both complete; Native-only success	100% → 100%	Resolved → Unresolved	Equal closure recall; repair regresses.

K ARTIFACT AND REPRODUCTION RECORD

This appendix records run-level reproduction coordinates and integrity checks; Appendix L separately maps evidence streams to source and data files. The artifact manifest records packaged input and script hashes, commands, environment metadata, instance IDs, random seeds, expected row counts, and per-stage attrition. The RQ4 package contains frozen action and score records plus sanitized

downstream outcome ledgers. Read-only aggregation reconstructs the benchmark–model matrix, deduplication, discordant pairs, controls, and aligned-cohort sensitivity from these processed records.

Execution and release checks. Declared failing tests ran in benchmark containers. The release scan checks scannable plain-text, gzip-compressed, and tar-gzip-compressed files for configured credential, identity, and absolute-home-path markers and rejects generated Python caches. Public trace summaries accompany the frozen actions and scores in the anonymous bundle.

Current RQ4 evidence entry points. Paths here are relative to the anonymous artifact root. The offline entry point, `reproduce_paper.sh`, reaggregates frozen scores in `data/rq4/closure/closure_per_instance.jsonl`. The current result is `rq4_budget_portfolio_complete_at_3` in `reports/reproduced_claims.json`: 764/2,289 to 944/2,289 complete pairs (33.38% to 41.24%; +7.86 points, 95% interval [5.80, 11.00]).

`reports/complete_at_3_per_model_defect.csv` exposes the 2,289-row panel, and `reports/aligned_controls_common_intersection_sensitivity.csv` records the common 1,965-pair gains of +5.39, +7.53, and +7.33 points. These reports use within-model aggregation across budgets; preserved legacy cross-model aggregates do not define the current endpoint.

`SHA256SUMS` binds bundled inputs, scripts, and reports; panel rows distinguish per-pair action and source-record hashes from global archive and ledger hashes. This replay reaggregates processed scores; it does not regenerate Native, recollect traces, or independently validate gold mapping or repair execution. A separate offline audit replays gold mapping and containment from retained patches, source structures, and actions, then reports the boundary correction and lexical sensitivity in Appendix N.3; it does not re-execute repairs. All four retained closure-run archives match their frozen compressed-file checksums. Complete archive scans recover the oversized-request response and final-binding evidence, but not the collector snapshot or original trace-output tree; multi-selector recollection remains underspecified (Appendix N.2). The 93 Lite reuse aliases retain copied collection-duration fields, not additional executions or a complete timing series; end-to-end latency and total cost remain unmeasured. `manifests/raw_archive_index.json` indexes raw-run references without implying that all raw inputs are included in the anonymous bundle.

K.1 Exact-Once and Resource Ledgers

The closure experiment reconciles every frozen-ledger stage against its expected Cartesian product. There are 852 stable logical benchmark rows and three models, yielding 2,556 model cells. Each cell records one fixed action and one isolated score. Five methods and three budgets across those model cells yield 38,340 rendered method–budget slots. The COMPLETE@3 rows then aggregate containment scores across budgets within each deduplicated model–defect pair. Table 36 verifies exact-once reconciliation and zero malformed-slot counts, while Table 37 summarizes the corresponding request and token records.

Table 36: Closure-experiment exact-once reconciliation counts across the frozen ledgers.

Ledger item	Expected	Reconciled
Stable logical benchmark rows	852	852
Unique benchmark–instance–model cells	2,556	2,556
Frozen action cells	2,556	2,556
Isolated score cells	2,556	2,556
Method–budget slots across model cells	38,340	38,340
Paired metric rows	108	108
Mechanism metric rows	27	27
Frozen per-budget arm summaries	45	45
Attrition records	351	351
Malformed or duplicate final keys	0	0

The physical ledger in Table 37 retains retries, sanitization, and superseded responses. The logical ledger establishes exact-once use in the canonical analysis. We report the canonical terminal fields used to compute token and request counts.

Table 37: Frozen model-transport and token ledger for the closure experiment. Counts summarize terminal metadata from successful responses in sanitized form.

Item	Count
Unique physical response request IDs	6,966
Successful request IDs	6,965
Current canonical technical failures	0
HTTP attempts including retries	6,977
Superseded response IDs retained for audit	178
Current unique request IDs	5,771
Logical manifest rows / recorded stage-cell uses	6,060 / 6,060
Prompt tokens	48,044,263
Completion tokens	6,746,789
Total tokens	54,791,052

K.2 Trace and Scoring Gates

The recorded benchmark-provided `FAIL_TO_PASS` input is submitted twice before any line enters TraceFill. As detailed in Appendix N.2, the available records do not establish that every selector in a multi-selector list was executed. A logical row qualifies only if it passes the frozen selector-resolution, expected-failure, non-interference, and same-exception checks. The intersection of the two source-line sets supplies candidate anchors. Trace instability is recorded as an explicit applicability outcome. Table 38 records each gate against its input population and attrition code.

Table 38: RQ4 gate semantics and their recorded attrition.

Gate	Input population	Removed	Recorded interpretation
Owner-valid roster	1,100 logical rows	3	Permanent benchmark-owner exclusions before trace collection.
Repeat-consistent trace	1,097 owner-valid rows	245	Excluded from closure scoring; downstream evaluation uses Native fallback.
Nonempty canonical Python closure	852 stable rows	31	Uniform closure-scoring denominator across models, arms, and budgets.
Structural control availability	2,289 model–defect pairs	21 / 22 / 288	Same-file / Anchor-count / Exact-LoC exclusions under all-three-cap eligibility.
Pair-symmetric patch availability	1,004 unique instances per model	117, 67, 1	For DeepSeek, Qwen, and GPT, respectively; pair-symmetric failures score zero.

Gold construction occurs in an isolated scoring stage after the generation freeze. Table 39 binds that separation to the immutable run identifiers and result ledgers used by each claim family.

Table 39: Core reproducibility coordinates. Hash prefixes identify immutable local artifacts; the supplementary manifest records full hashes, commands, schemas, environments, and per-instance ledgers.

Claim	Frozen artifact	Scope
RQ1 boundaries	RQ1 mechanical reanalysis (`8e2d406b6b9e`)	Read-only reconciliation of already scored outcomes and mechanical closure taxonomy.
RQ2 primary	Repeated exact-test `results_v1/summary` (`cd0256db7287`)	Repeated exact FAIL_TO_PASS result over serialized actions and controls.
RQ2 breadth / structure	`phase1_b2` (`7d3bc45f27d2`); `ablation2_b1/b2_a0.9`	Frozen candidate ceiling, budgeted selector, and 200-seed matched-permutation summaries.
RQ2 supporting audits / interventions	Valid-subset recomputation (`52ebf8c35656`); frozen intervention ledgers	Paired crash-selection audit, sham-controlled AST response, and directed-predicate certification.
CFRR diagnostic	Canonical 117-row ledger (`3fa6566265ea`)	Native realization and gold-aware minimum-union reach/fit counts over the frozen universe.
RQ4 closure	`wp_a_3x3_20260817T075609Z`	Three benchmarks, three models, three controls, three budgets; gold-isolated closure scoring.
RQ4 components / stability	`iclr2027_tracefill_component_ablation_20260823`; `iclr2027_stability_filter_ablation_20260823`	Frozen offline variants, paired tests, full-roster trace variation, and exact reconciliation to the closure run.
RQ4 alignment	Read-only instance join	Logical/physical deduplication, attrition, and 763-instance sensitivity from frozen outputs.

L AUDITED CODE AND DATA INDEX

Where Appendix K gives run coordinates and integrity checks, this index maps each reported result family to its source modules, fixed input ledger, and presentation artifact. Per-instance records remain in the supplementary ledger, which retains full instance IDs, hashes, and execution metadata.

Audit protocol. We statically checked the Python and JavaScript sources in Table 40, parsed the packaged machine-readable ledgers in Table 41, and traced each reported matrix and figure to a local artifact. Reaggregation reads the processed ledgers and preserves all frozen inputs.

Table 40: Audited active source map. Source units are grouped by evidence stream; each group is paired with the frozen records named in Table 41.

Evidence stream	Audited source units	Contribution to the paper
Canonical data and rendering	`p0_line_recall.py`, `code_graph.py`, `egl_makeorbreak.py`, `p7_crossfile_exec.py`	Canonical pre-fix line mapping, source normalization, repository execution interface, and rendered-footprint accounting.
RQ1 mechanical audit	`p0_measurement_audit/` reanalysis and measurement-audit scripts; `rq1/` dimension code	Mechanical taxonomy, denominator reconciliation, and complete-context boundary summaries.
RQ2 execution mechanisms	`rq2/` crash-coverage code; `rq3/` execution, SBFL, intervention, and worked-example scripts	The six complementary direct-signal tests, their matched controls, and worked code views.
CFRR audit	`egl_pareto/`: canonicalization, candidates, reachability and selection analyses, ablations, miss types, and verification	One-expansion canonical evaluation, candidate reachability, minimum-union verification, and reachability–realization accounting.
Benchmark and Native base	`newbench/` construction and verified coverage; `agentless/` frozen baseline and vote artifacts	Frozen cohort construction, verified execution collection, and the preserved Native localization base.
RQ4 component analyses	`experiments/iclr2027_tracefill_component_ablation_20260823/`; `experiments/iclr2027_stability_filter_ablation_20260823/`	Gold-blind action rebuilding, component variants, paired scoring, trace-volatility accounting, and reconciliation.
RQ4 result materialization	`figures/` Vega, JavaScript, and Draw.io sources	Deterministic vector rendering of TraceFill, closure containment, and official-harness result matrices.

Table 41: Audited frozen data ledger. Each entry is a machine-readable local input or output behind a reported cohort, control, or endpoint.

Result family	Frozen ledger or presentation data	Auditable content
RQ1	`p0_measurement_audit/`: RQ1 reanalysis JSON, mechanical taxonomy CSV, and canonical manifest	The 117 evaluable Native observations, mechanical closure taxonomy, and reconciliation provenance.
RQ2 exact and support	`fresh219_roster_2026-07-20.jsonl`, `rq3_subsets.json`, `p0_rq2_valid_subset_recompute.json`	Frozen roster, exact-test eligibility, mechanism subsets, and the repeated-trace evidence ledger.
Execution and benchmark data	`egl_cov_cache.json`, `newbench/{audit_crash_half, benchmark2, verified_cov_cache}.json`	Cached execution evidence, benchmark membership, and independently frozen verified coverage.
CFRR and structural controls	`egl_pareto/`: reachability and selection results, two $\alpha = .9$ ablations, miss types, and manifest	Reachability ceilings, realized selections, matched-control summaries, miss types, and manifest metadata.
RQ4 closure containment	Immutable closure run; `complete_at_3_per_model_defect.csv`; `complete_by_budget_dataset_model.csv`	The 2,289-row model–defect panel with per-row action/score hashes and all 27 descriptive benchmark–model results.
RQ4 components / stability	CSV, JSON, and Markdown result ledgers for the component and stability experiments, plus the two Vega-Lite specifications	Pooled ablations, exact-line sensitivity, matched controls, full-roster trace variation, and containment under alternative trace-set choices.
Reproduction metadata	Claim registry, canonical metadata audit, and Table 39	Claim-to-artifact mapping, input/code hashes, bibliography metadata, and deterministic reaggregation coordinates.

Reproducibility record. These source and data mappings support joint inspection of the reaggregation code, processed records, and reported statistics.

M LIMITATIONS AND SCOPE

The paper reports a reference-patch closure surrogate, not the set of all correct repairs. The two development cohorts were repeatedly used during diagnosis. RQ1 covers single-file, on-path defects, and CFRR's stage distribution depends on this development cohort and its fixed candidate family. The 26-case diagnostic headroom does not predict recovery by TraceFill, which uses a larger cohort and one Native base per defect and model. Repository comparisons are descriptive: project architecture, patch conventions, and the curated defect mix are intertwined.

RQ2 tests specified policies using exact traces and whole-module proxy coverage in experiment-specific cohorts; the earlier 17-instance audit is a mechanism precursor. The results do not bound every gold-blind selector on an expanded pool. Experiment 5 groups sites by candidate provenance, not reference-closure membership; its response-rate comparison establishes neither closure-membership discrimination nor response-rate equivalence. Comparisons with AutoCodeRover, SieveFL, SWER-ank, and RepoMem distinguish mechanisms, scored objects, and budgets; no head-to-head evaluation establishes TraceFill's relative effectiveness against these systems. The fail–pass subtraction probe is not a comparison of spectrum-based suspiciousness rankings under TraceFill's residual caps.

RQ4 actions and runs were frozen before gold scoring, but COMPLETE@3 was defined through post-freeze reaggregation. The study remains development-informed, and excluding development-overlap defects does not create a fresh confirmation set. Post-hoc controls and retrospective component ablations do not isolate an execution-specific causal effect; aligning the common cohort addresses denominator differences, not causal identification. The small allocation contrast gives limited evidence about scan-and-skip. Trace-set comparisons condition on the two-run gate and show no additional containment benefit from intersection. They neither establish semantic-noise removal nor evaluate a single-run admission policy. Post-hoc stratification shows that most of the net resolved-count increase occurs in cases with empty Native contexts, many arising from output-format and parser incompatibility. The observed results do not measure performance after correcting that interface. The repair comparison lacks a matched-extra-context arm and therefore does not identify an execution-specific selection effect or a causal closure-to-repair mechanism. KEEP guarantees source containment, not repair monotonicity.

Benchmark rows overlap, and model rows reuse the same physical defects, so these observations are not independent. Fixed-panel summaries deduplicate instance IDs, and no cross-model significance test is reported. The three-model panel limits generalization beyond the evaluated systems.

Appendix K records the release checks: exact checksum-manifest coverage and scans of the supported package formats for configured credential, identity, absolute-home-path, and Python-cache markers. These checks apply to the submitted bundle.

Future work could replace the provenance-tier order with a learned estimate of $p_\theta(\ell \in G_i)$ from semantic, execution, and dependency features. Such a model would require repository-disjoint training labels while preserving KEEP, each declared budget, and the gold firewall at inference. Alternative-patch and cross-file/off-path evaluations would test the formulation beyond the present cohorts.

N RECORDED INPUT CONSTRUCTION AND CANONICAL MAPPING

This appendix specifies the recorded input branches, Native construction, and reference-line mapping for the frozen closure-containment matrix and its downstream repair-context handoff. It distinguishes generated contexts, prepared requests, and scored outcomes.

N.1 RECORDED REPAIR-CONTEXT BRANCHES

The request-side context audit covers 1,004 physical defects. Of these, 789 have stable traces and 215 fail the trace gate. The 763-defect closure panel is a subset of the stable group; the remaining 26 stable defects are outside that panel after the frozen empty-closure exclusion. The gate-failure group and the closure panel are disjoint. Native generation in the archived closure-evaluation procedure is scheduled only for instances whose trace status is `STABLE`. Thus, the 215 gate-failure instances do not have an attempted Native generation in that procedure. The repair handoff records both context files as empty

for these instances, with the SHA256 of an empty byte string and null Native-action/source-structure bindings. These records do not establish that an independently generated, pre-gate semantic context would also have been empty.

In this run-specific sense, *Native fallback* for a gate failure is reuse of the recorded empty request-side Native context.

Dataset reuse makes the 215 physical gate failures correspond to 245 logical dataset–instance entries, or 490 two-arm entries per model; these are not 490 independent defects. For each of the 215 physical cases, each generator's recorded queue maps both arms to the same prepared request identity within a model; all corresponding source fields are empty, while the issue and repair instructions remain present. Prepared requests need not yield a successful generation: the paired technical-failure masks in this subgroup contain 16 DeepSeek, two Qwen, and zero GPT physical pairs, which receive zero in the full-roster endpoint. This technical-failure convention is distinct from the empty-context branch.

N.2 Native generation and input prerequisites

The Native base used for closure evaluation is generated by a trace-gated, three-stage localization pipeline. Only instances with a stable trace terminal enter this procedure. File localization receives the issue description and the repository structure at the base commit, and requests at most five file paths ordered by importance. Parsed paths are corrected against the repository file list; the retained Python paths and their order define the found-file list and rank. Related-symbol localization receives the issue and skeletons of those files and requests classes, functions, and variables. Edit-location localization receives the issue and numbered source context around the related locations and requests edit locations. It is skipped when that context is empty.

Submitted requests and one frozen base. For each model alias—`deepseek-v3.2`, `qwen3.5-plus`, and `gpt-5.3-codex`—each reached stage submits one user message, `temperature=0`, `n=1`, and `max_tokens=300`, with no model tools or functions. There is no multi-sample vote or best-of-N selection. These settings were checked in all 5,771 distinct submitted request records referenced by the frozen logical ledger; their canonical request hashes match. Technical retries and reuse of identical requests are distinct from scientific sampling. Parsed edit locations are converted to one frozen anchor set; location-conversion exceptions are skipped. The same base and its hash are reused for all arms and all three residual caps. The total source-line allowance remains the cell's own Native LoC plus $b \in \{51, 170, 414\}$, rather than a common total token allowance across models.

Recorded failures and size. Localization-output parsing failures produce an empty edit set under a uniform rule; pending requests and technical failures follow the recorded retry procedure. A complete Native terminal can still map to an empty anchor set. Table 42 reports availability and size using the 2,289 canonical endpoint action hashes. All 2,289 Native terminals were recovered from the archive by their bound Native-action hashes, so missing local files are not classified as empty bases.

Table 42: Native base availability and rendered size on the canonical 763 defects per model. LoC is the unique source-line union after one final ± 10 expansion, including zero-sized bases, before adding residual capacity. Model names denote recorded API aliases.

Model	Empty / 763	Empty (%)	Mean LoC	Median LoC	Max LoC
DeepSeek-V3.2	562	73.66	195.64	0	9,972
Qwen3.5-Plus	8	1.05	232.69	101	4,110
GPT-5.3-Codex	171	22.41	272.53	124	4,592

The 562 empty DeepSeek bases comprise 561 stage-one and one stage-two parsing failures. GPT has 133 stage-one, five stage-two, and 32 stage-three parsing failures, plus one complete terminal whose locations map to an empty base. Qwen has two stage-one, one stage-two, and one stage-three parsing failures, two empty stage-three contexts, and two complete terminals mapping to empty bases. These categories describe this frozen configuration; they do not identify one causal failure mechanism.

The request parameters specify what was submitted. They do not establish identical effective decoding limits across aliases. All canonical stage-one response contents are nonempty; 762 of 763 DeepSeek

responses report `finish_reason=length`, whereas the GPT and Qwen responses report `stop`. The parser reads the first choice's content without requiring a particular finish reason. API-reported Qwen completion usage can exceed the submitted 300-token value. Thus, an empty rendered base is distinct from an empty API response, and the submitted limit is not a verified cross-provider effective cap.

A technical amendment frozen before gold scoring specifies a 240,000-byte prompt limit, with equal prefix/suffix retention around a fixed middle marker. Fifteen current line-localization requests exceed this limit: three DeepSeek, six Qwen, and six GPT, up to 974,746 UTF-8 bytes. Each has a retained HTTP-200 response with matching request-specification hash and `finish_reason=stop`; parser replay using hash-recorded local helpers reproduces its retained complete Native terminal, which binds to the reported closure endpoint. Two additional over-limit specifications were superseded: a DeepSeek HTTP-400 request was replaced by a 240,000-byte request, while a Qwen specification was replaced during local sanitation without a retained HTTP terminal for the old identity. These records establish the current exceptions and their final use, but neither uniform enforcement of the amendment nor provider-internal truncation behavior. This replay does not independently certify the historical helper snapshot.

Configuration and test-input coordinates. Within the immutable closure-evaluation run in Table 39, the generation entry points are `scripts/build_stage1_api_requests.py`, `scripts/build_agentless_downstream.py`, and `scripts/build_tracefill_actions.py`. Request identities are in `final_analysis/logical_request_ledger.jsonl`, with retained bodies under `queues/api_pending/`; these records do not independently certify a provider's model snapshot. Frozen trace queues declare multiple `FAIL_TO_PASS` selectors for 388/1,004 physical instances and 416/1,097 logical instances, including 307/789 stable physical instances and 301/763 closure-evaluable defects (39.45%). Counts use the final handoff and its reuse map. The current Lite source-roster file differs from its queue-recorded file hash, but all 300 selector lists agree with the frozen queue. The available worker passes the input record to the collector twice. Complete scans of the four checksum-verified run archives do not recover the collector snapshot or the original `outputs/` trace tree identified by the host indices. The declared-list distribution therefore does not establish the multi-selector execution or stopping rule, nor verify completion of every listed selector.

N.3 Frozen Reference-Line Mapping for Closure Evaluation

The original `score_actions_isolated.py` implementation is bound to all four host generation freezes and 2,556 pre-attrition model cells. An offline audit matches the archived patches and source structures to those bindings and exactly reproduces all 38,340 method–budget rows for gold size, rendered cost, hit count, graded recall, file exposure, availability, and containment. Original records remain immutable. A versioned boundary-state correction is evaluated separately from the lexical-anchor sensitivity below; neither changes the frozen actions. This audit concerns the closure matrix, not identical behavior of every historical diagnostic helper.

Hunk parsing and lexical cleaning. A `+++ b/` header sets the file key. At a hunk header, the old-side cursor starts at the old hunk start. Context lines advance the cursor. A deleted line contributes its current old-side position and then advances the cursor; each added line contributes $\{\mathrm{old}-1, \mathrm{old}\}$ after removing positions below one, without advancing the cursor. Consequently, replacement hunks can include an adjacent original context line as well as the deleted line. This is an edit-location mapping, not a mapping of added code to post-fix line numbers. Only `.py` paths present under the same key in the frozen source structure are retained. The cleaner removes out-of-range positions, stripped blank lines, and comment-only lines. It then takes the substring before the first `#`, trims surrounding whitespace, and discards the position if the resulting nonempty substring consists solely of `()[]{},:;\`. This is lexical filtering, with no AST analysis or relocation of discarded anchors to neighboring code. The final base-plus-addition anchors receive one inclusive ± 10 expansion, clipped to the original file bounds and deduplicated as $(\mathrm{file}, \mathrm{line})$ pairs. The gold line set itself is not expanded.

Patch-form boundaries. Table 43 records the original parser. These rules need not preserve every hunk or changed-language region. If lexical cleaning removes all anchors of an addition hunk while

another hunk retains mapped lines, G_i remains nonempty. A context can satisfy $G_i \subseteq W_i(A)$ while omitting that modification site. The audit below separates partial hunk omission, entirely empty mapped closures, and actual changes to frozen-context scores.

Table 43: Patch-form handling in the frozen closure-evaluation mapper, before lexical cleaning. The rules describe the recorded implementation.

Patch form	Pre-fix mapping rule and boundary
Replacement	Deleted old-side positions plus $\{\mathrm{old}-1, \mathrm{old}\}$ for additions; may include an adjacent original context line.
Addition-only hunk	Uses the same two old-side anchors, without relocation. A zero-context hunk starting at old position k contributes $\{k-1, k\}$, discarding positions below one.
Deletion within a file	With a `+++ b/` file key, deleted lines contribute their old positions, then undergo cleaning.
New file	Standard new-file hunks start at old position zero; no virtual pre-fix file is constructed. Paths absent from the frozen source structure are excluded.
Rename	Rename metadata is not interpreted and the new path is not mapped back to the old path. A new key absent from the source structure is excluded.
Whole-file deletion	A `+++ /dev/null` header does not reset the current key. Its hunk can lack a key or retain a preceding file's key.
Mixed languages	Only `.py` paths present under the same key in the frozen source structure are retained.

Boundary-state correction. Five of the 789 physical records contain whole-file deletion hunks with stale raw file attribution. Resetting state at actual file headers, with no carry-over after `+++ /dev/null`, changes retained gold in two cases: `tox-dev__tox-3388` loses 69 spurious coordinates (109 $\rightarrow$ 40 lines), and `pydata__xarray-9636` loses 33 (386 $\rightarrow$ 353). Source text resembling a header inside a hunk is not treated as a boundary. The correction changes some hit counts and graded recalls, but no Complete label, availability flag, rendered cost, or eligible denominator across the frozen matrix. The 763-defect panel and all reported Complete comparisons therefore remain unchanged. Existing trace summaries also reproduce both cases' original candidate diagnostics: reachability and all three capacity-fit labels remain unchanged, while tox's minimum added LoC changes from 354/309/187 to 271/226/184 for DeepSeek/Qwen/GPT; xarray remains unreachable. Corrected scores retain links to the original records. This fixes cross-file state carry-over without introducing old-path mapping for deletion, rename normalization, or virtual pre-fix files.

Hunk-level omission audit. The audit distinguishes pure-addition hunks (added lines and no deleted lines) from mixed or deletion hunks. Across 852 logical records, 3,541 patch hunks include 1,106 pure-addition hunks; 872 have original, in-bounds Python anchors. After physical reuse, the corresponding counts are 3,453, 1,082, and 848. Table 44 separates hunk and instance denominators. In the fixed 763-defect panel, 55/840 eligible pure-addition hunks disappear after lexical cleaning (6.55%), affecting 51/373 defects with such hunks (13.67%; 51/763 overall, 6.68%). All 51 still have nonempty mapped gold. Another 25 non-pure-addition hunks across 19 physical defects lose all in-bounds Python anchors; these are recorded but are outside the pure-addition restoration rule.

Table 44: Pure-addition hunk omissions. Hunk denominators require original in-bounds Python anchors. The instance denominator in column four counts defects with at least one such hunk; the last column counts affected defects whose original gold remains nonempty.

Scope	Records	Hunks lost / eligible	Affected defects / eligible	Nonempty affected
Logical	852	67 / 872	63 / 403	53
Physical	789	63 / 848	59 / 381	51
Fixed panel	763	55 / 840	51 / 373	51

Fixed-context metric sensitivity. For each entirely cleaned pure-addition hunk, a declared sensitivity restores only its original in-bounds Python anchors. It neither relocates anchors nor changes any context or cap, and is distinct from correcting file state. On the fixed 763 defects, Native/TraceFill Complete@3 counts change from $764/944$ to $763/941$ out of 2,289 pairs: 33.33% versus 41.11%, a gain of 7.78 points instead of 7.86. Fixed-panel model counts become $126 \rightarrow 189$, $398 \rightarrow 447$, and $239 \rightarrow 305$ for DeepSeek, Qwen, and GPT. Reapplying the same empty-gold exclusion admits eight additional physical defects, yielding $775 \rightarrow 953$ out of 2,313 pairs on 771 defects (33.51%

to 41.20%, +7.70 points). Restoration with and without the boundary correction yields the same Complete labels. These are descriptive sensitivity results, not replacements for the lexical metric or new significance tests.

On the original 1,965 control-matched pairs, the restored metric gives 785 complete TraceFill contexts versus 635 Native, 678 Same-file, 639 Anchor-count, and 643 Exact-LoC contexts under COMPLETE@3: advantages of 7.63, 5.45, 7.43, and 7.23 points. Under Complete(414), the respective counts are 785 versus 635, 663, 639, and 639. Matching identities, availability, and source costs remain fixed. A removed anchor need not be absent from a rendered window, and absence need not change an already-incomplete label; the audit retains per-context coverage and label transitions rather than treating every omission as a scoring error. This sensitivity does not address alternative correct patches or recover unsupported path/language forms.

Empty mapped closures and reuse. The scorer requires a nonempty gold set for contains-all success; an empty set is not a vacuous success. The frozen post-gold attrition rule excludes an instance when all three models record zero gold files and zero gold lines, uniformly across arms and budgets. It removes 31 of 852 logical instances (six Verified, 17 Live Lite, and eight Lite), leaving 821. These 31 logical records contain 29 distinct instance IDs. After the frozen cross-dataset reuse map selects the shared physical record, 26 of 789 physical instances have an empty mapped closure, leaving the reported 763-defect panel. Empty mapped closure does not mean that the reference patch contains no Python modification. Two archived examples illustrate the distinction. In `matplotlib-23476`, the addition in `lib/matplotlib/figure.py` maps to original lines 3025 and 3026, respectively a blank line and a comment. In `sympy-20590`, the addition in `sympy/core/_print_helpers.py` maps to original lines 19 and 20, again blank/comment lines. Both pairs are removed by the frozen cleaner.

The archived Lite source structures for `astropy-14365`, `django-13401`, and `django-13658` are empty objects. Their Verified counterparts have recorded gold-line counts of four, six, and two; the reuse map selects those Verified records for the pooled analysis. Their upstream cause, and the causes of the other empty logical records, have not all been determined. The examples do not constitute a complete exclusion-reason distribution.